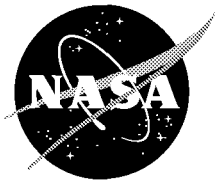
NASA/CP—2000–209890

SEL–99–002
Software Engineering Laboratory Series



Proceedings of the Twenty-Fourth Annual Software Engineering Workshop

Compiled by:
Goddard Space Flight Center

Proceedings of a workshop held
at the Goddard Space Flight Center
Greenbelt, Maryland
December 1–2, 1999

National Aeronautics and
Space Administration

Goddard Space Flight Center
Greenbelt, Maryland 20771

March 2000

The NASA STI Program Office … in Profile

Since its founding, NASA has been dedicated to the advancement of aeronautics and space science. The NASA Scientific and Technical Information (STI) Program Office plays a key part in helping NASA maintain this important role.

The NASA STI Program Office is operated by Langley Research Center, the lead center for NASA's scientific and technical information. The NASA STI Program Office provides access to the NASA STI Database, the largest collection of aeronautical and space science STI in the world. The Program Office is also NASA's institutional mechanism for disseminating the results of its research and development activities. These results are published by NASA in the NASA STI Report Series, which includes the following report types:

- TECHNICAL PUBLICATION. Reports of completed research or a major significant phase of research that present the results of NASA programs and include extensive data or theoretical analysis. Includes compilations of significant scientific and technical data and information deemed to be of continuing reference value. NASA's counterpart of peer-reviewed formal professional papers but has less stringent limitations on manuscript length and extent of graphic presentations.
- TECHNICAL MEMORANDUM. Scientific and technical findings that are preliminary or of specialized interest, e.g., quick release reports, working papers, and bibliographies that contain minimal annotation. Does not contain extensive analysis.
- CONTRACTOR REPORT. Scientific and technical findings by NASA-sponsored contractors and grantees.
- CONFERENCE PUBLICATION. Collected papers from scientific and technical conferences, symposia, seminars, or other meetings sponsored or cosponsored by NASA.
- SPECIAL PUBLICATION. Scientific, technical, or historical information from NASA programs, projects, and mission, often concerned with subjects having substantial public interest.
- TECHNICAL TRANSLATION. English-language translations of foreign scientific and technical material pertinent to NASA's mission.

Specialized services that complement the STI Program Office's diverse offerings include creating custom thesauri, building customized databases, organizing and publishing research results . . . even providing videos.

For more information about the NASA STI Program Office, see the following:

- Access the NASA STI Program Home Page at http://www.sti.nasa.gov/STI-homepage.html
- E-mail your question via the Internet to help@sti.nasa.gov
- Fax your question to the NASA Access Help Desk at (301) 621-0134
- Telephone the NASA Access Help Desk at (301) 621-0390
- Write to:
  NASA Access Help Desk
  NASA Center for AeroSpace Information
  7121 Standard Drive
  Hanover, MD 21076–1320

The views and findings expressed herein are those of the authors and presenters and do not necessarily represent the views, estimates, or policies of the SEL. All material herein is reprinted as submitted by authors and presenters, who are solely responsible for compliance with any relevant copyright, patent, or other proprietary restrictions.

Available from:

NASA Center for AeroSpace Information
7121 Standard Drive
Hanover, MD 21076–1320
Price Code: A17

National Technical Information Service
5285 Port Royal Road
Springfield, VA 22161
Price Code: A10

Proceedings of the Twenty-Fourth Annual Software Engineering Workshop

December 1–2, 1999

GODDARD SPACE FLIGHT CENTER

Greenbelt, Maryland

FOREWORD

The Software Engineering Laboratory (SEL) is an organization sponsored by the National Aeronautics and Space Administration/Goddard Space Flight Center (NASA/GSFC) and created to investigate the effectiveness of software engineering technologies when applied to the development of applications software. The SEL was created in 1976 and has three primary organizational members:

NASA/GSFC, Information Systems Center
The University of Maryland, Department of Computer Science
Computer Sciences Corporation, Development and Sustaining Engineering Organization

The goals of the SEL are (1) to understand the software development process in the GSFC environment; (2) to measure the effects of various methodologies, tools, and models on this process; and (3) to identify and then to apply successful development practices. The activities, findings, and recommendations of the SEL are recorded in the Software Engineering Laboratory Series, a continuing series of reports that includes this document.

Documents from the Software Engineering Laboratory Series can be obtained via the SEL homepage at:

http://sel.gsfc.nasa.gov/

or by writing to:

Systems Integration and Engineering Branch
Code 581
Goddard Space Flight Center
Greenbelt, Maryland 20771

CONTENTS

Materials for each session include the viewgraphs presented at the workshop and a supporting paper submitted for inclusion in these *Proceedings*.

Session 1: The International Influence of the Software Engineering Laboratory

Rini van Solingen, Eindhoven University of Technology and Tokheim/Netherlands

Dieter Rombach, Fraunhofer Institute for Experimental Software Engineering

Frank Houdek, Daimler-Benz/Germany

Experiences in Using the Goal/Question/Metric Paradigm

Rini van Solingen

Tokheim, The Netherlands
Eindhoven University of Technology, The Netherlands

R.v.Solingen@tm.tue.nl
Http://www.gqm.nl/

Abstract

Tokheim, a company that provides products and services for the retail petroleum market, applies the Goal/Question/Metric paradigm to support their software development projects in their central development site in the Netherlands since 1994. Many experiences have been gathered during these projects. Experiences includes knowledge on software development topics, but also on practical GQM application in industry.

The presentation will address a selection of experiences, lessons learned and measurement examples collected during the past years.

GQM experiences published in a book

These experiences have also been published recently in the McGraw-Hill book: 'The Goal/Question/Metric method: A practical guide for quality improvement of software development' by Rini van Solingen and Egon Berghout. ISBN 0-07-709553-7.

This book contains practical procedures for GQM application in industry and consists for over 50% of practical results and documents from GQM application in four Tokheim projects.

Foreword by Professor Victor R. Basili to the GQM book

The original ideas for the Goal Question Metric Paradigm came from the need to solve a practical problem back in the late 1970s. How do you decide what you need to measure in order to achieve your goals? We (Dr. David Weiss and I) faced the problem when trying to understand the types of changes (modifications and defects) being made to a set of flight dynamics projects at NASA Goddard Space Flight Center. Was there a pattern to the changes? If we understood them could we anticipate them and possibly improve the development processes to deal with them? At the same time, we were trying to use change data to evaluate the effects of applying the Software Cost Reduction methodology on the A-7 project requirements document at the Naval Research Laboratory.

Writing goals allowed us to focus on the important issues. Defining questions allowed us to make the goals more specific and suggested the metrics that were relevant to the goals. The resulting GQM lattice allowed us to see the full relationship between goals and metrics, determine what goals and metrics were missing or inconsistent, and provide a context for interpreting the data after it was collected. It permitted us to maximize the set of goals for a particular data set and minimize the data required by recognizing where one metric could be substituted for another.

The process established the way we did measurement in the Software Engineering Laboratory at Goddard Space Flight Center, and has evolved over time, based upon use. Expansion involved the application to other areas of measurement (such as effort, schedule, process conformance), the development of the goal templates, the development of support processes, the formalization of the questions into models, and the embedding of measurement in an evolutionary feedback loop, the Quality Improvement Process and the Experience Factory Organization. Professor Dieter Rombach was a major contributor to this expansion.

The GQM paradigm represents a practical approach for bounding the measurement problem. It provides an organization with a great deal of flexibility, allowing it to focus its measurement program on its own particular needs and culture. It is based upon two basic assumptions (1) that a measurement program should not be 'metrics-based' but 'goal-based' and (2) that the definition of goals and measures need to be tailored to the individual organization. However, these assumptions make the process more difficult than just offering people a "collection of metrics" or a standard predefined set of goals and metrics. It requires that the organization make explicit its own goals and processes.

In this book, Rini van Solingen and Egon Berghout provide the reader with an excellent and comprehensive synthesis of the GQM concepts, packaged with the support necessary for building an effective measurement program. It provides more than the GQM, but describes it in the philosophy of the Quality Improvement Paradigm and the Experience Factory Organization. Based upon experience, they have organized the approach in a step-by-step set of procedures, offering experience-based heuristics that I recognize as effective. They have captured the best ideas and offer them in a straightforward manner. In reading this book, I found myself constantly nodding in agreement, finding many ideas I had not articulated as well. They offer several examples that can be used as templates for those who wish to have a standard set of goals and metrics as an initial iteration.

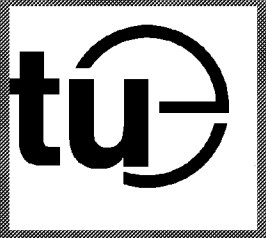
If you work on a measurement program, you should keep this book with you as the definitive reference for ideas and procedures.

Professor Victor R. Basili
University of Maryland
and
Fraunhofer Center for Experimental Software Engineering, Maryland

About the presenter

Rini van Solingen (M.Sc.) has been working as a senior software quality engineer at Tokheim and as a research fellow at Eindhoven University of Technology, since 1994. During this period he worked on all Tokheim GQM projects and performed research on software process improvement and measurement. He has published over 50 publications in international journals and conference proceedings. He is a member of the IEEE Computer society and is an active reviewer for IEEE Software.

Experiences in Using the Goal/Question/Metric Paradigm



by

Rini van Solingen

TOKHEIM® QUALITY

Eindhoven University of Technology

Tokheim

The Netherlands

Contents

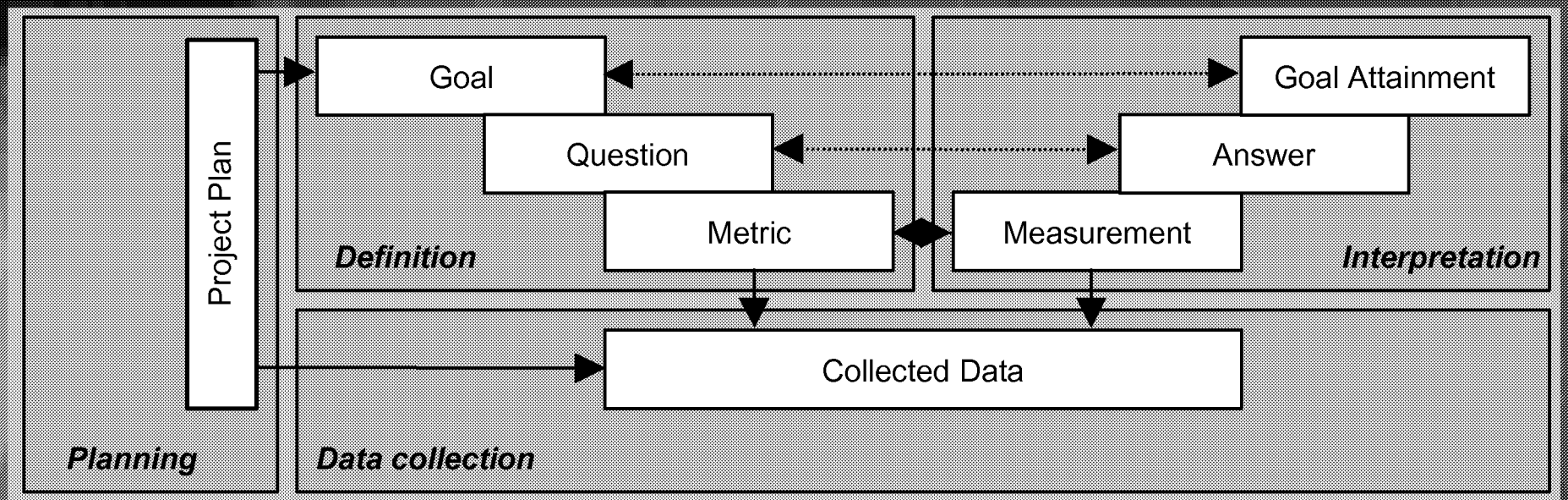
- Tokheim
- The Goal/Question/Metric method
- Examples from practice:
  - Reliability
  - Reuse
  - Interrupts
- Lessons learned
- Conclusions

Tokheim

TOKHEIM® QUALITY

- Petroleum retail systems and services:
  - Fuel dispensers
  - Payment and cashing systems
  - Petrol station construction
  - Maintenance and refurbishing services
- 4.800 employees in 145 countries
- Annual revenue: US$ 700 million
- GQM experiences started in former Schlumberger RPS, September 1994

The GQM method

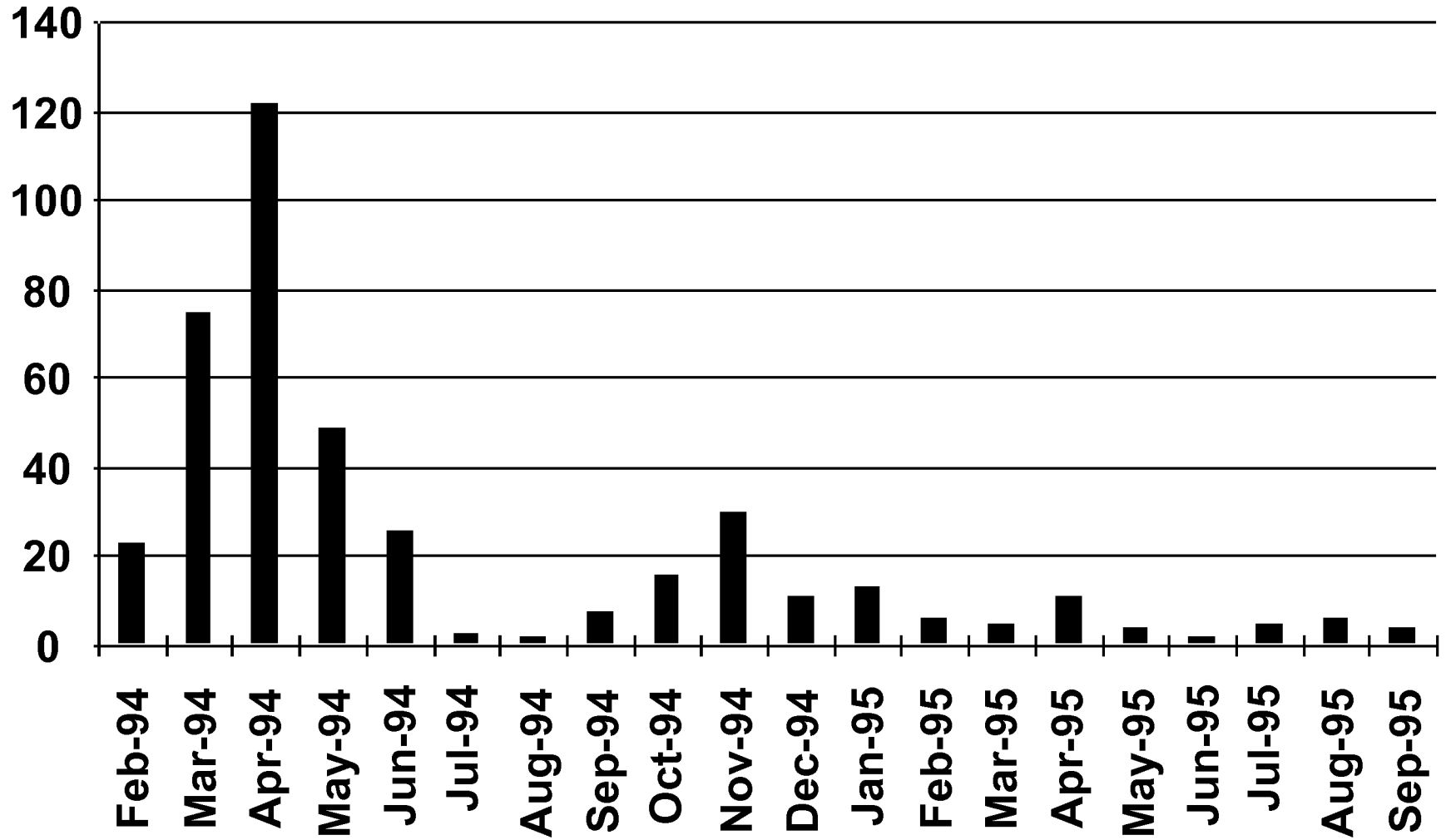


- **Four phases:**
  - **Planning: preparation**
  - **Definition: specification of GQM**
  - **Data collection: measurement**
  - **Interpretation: feedback sessions**

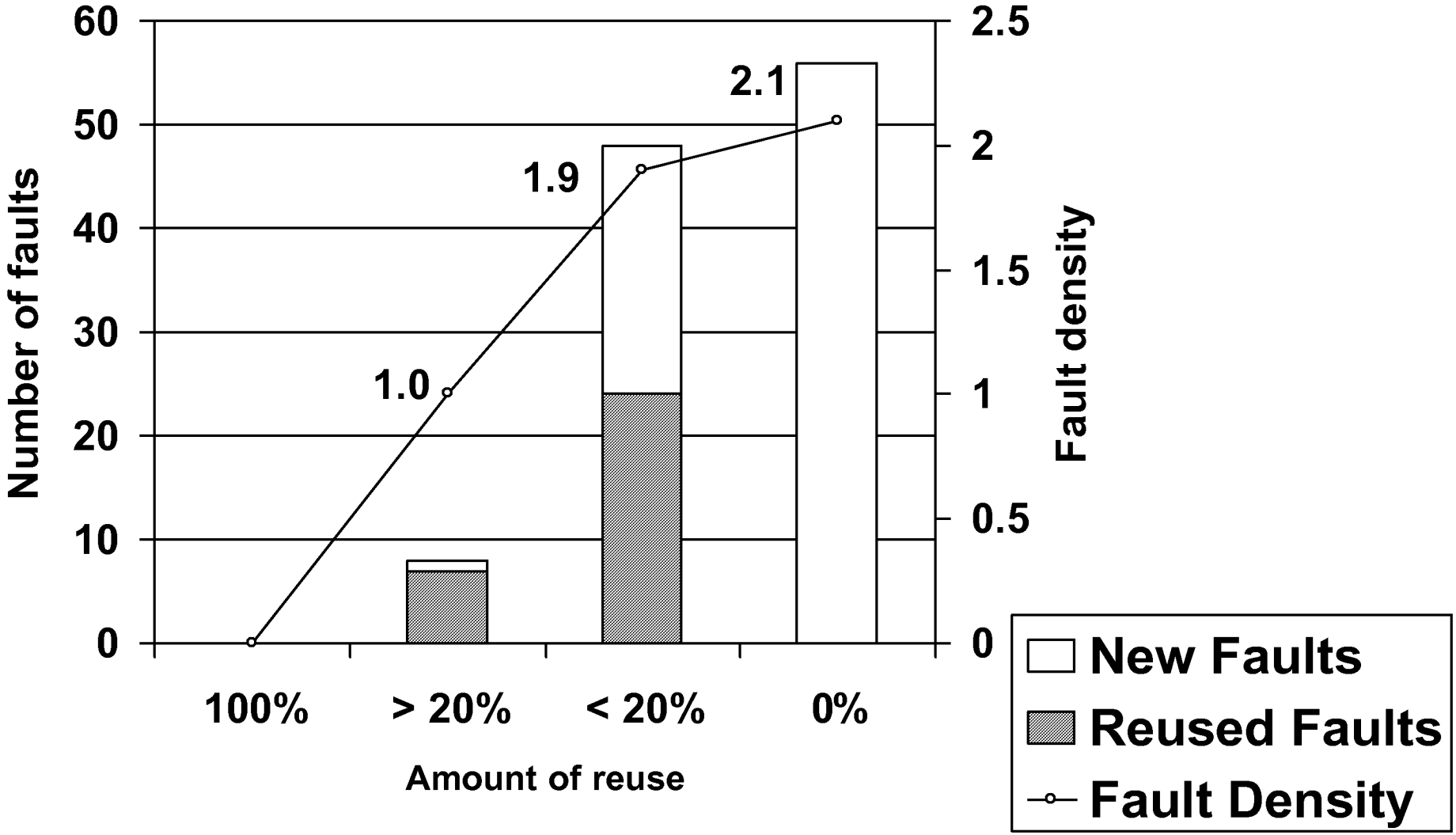
GQM fits to industrial needs

- Goal-orientation suits industry
- Support for multiple goals
- Focusing to relevant topics
- Solving relevant problems
- Both product and process related
- Limiting of measurement investment
- Supporting projects and teams, not organisations as a whole

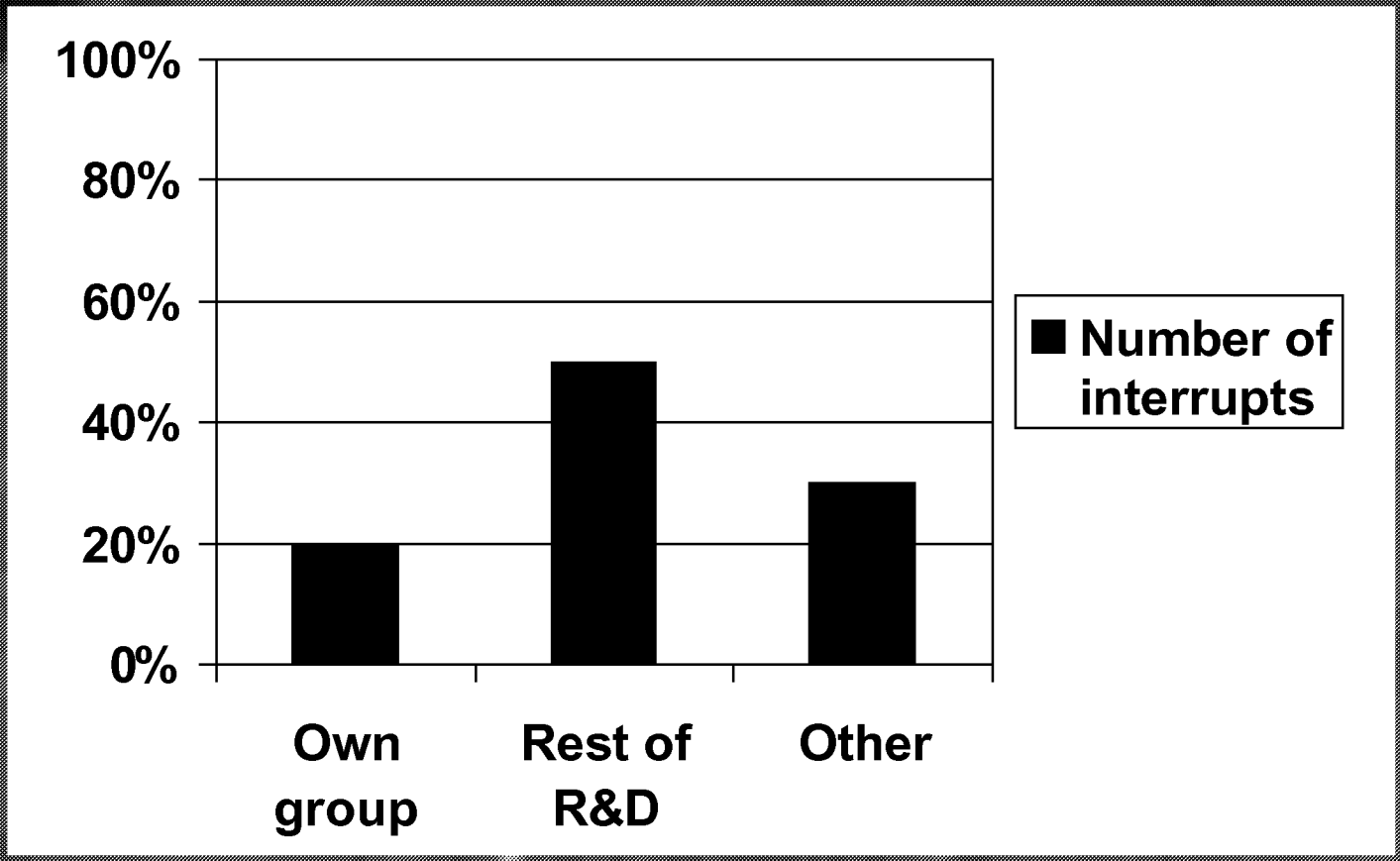
Example: Reliability



Example: Reuse



Example: Interrupts



- Own Group is Number 1 initiator
- 70% of interrupts originate within R&D
- Decrease of interrupts 30-50% in 3 months (5% gain)

Lessons learned

- GQM supports industrial learning
- Learning rarely an explicit goal in industrial environments
- Cost are limited: <1% of project efforts
- Benefits exceed the costs largely
- ROI > 10, payback time < 3 months
- GQM support group necessary and can be incorporated in QA department

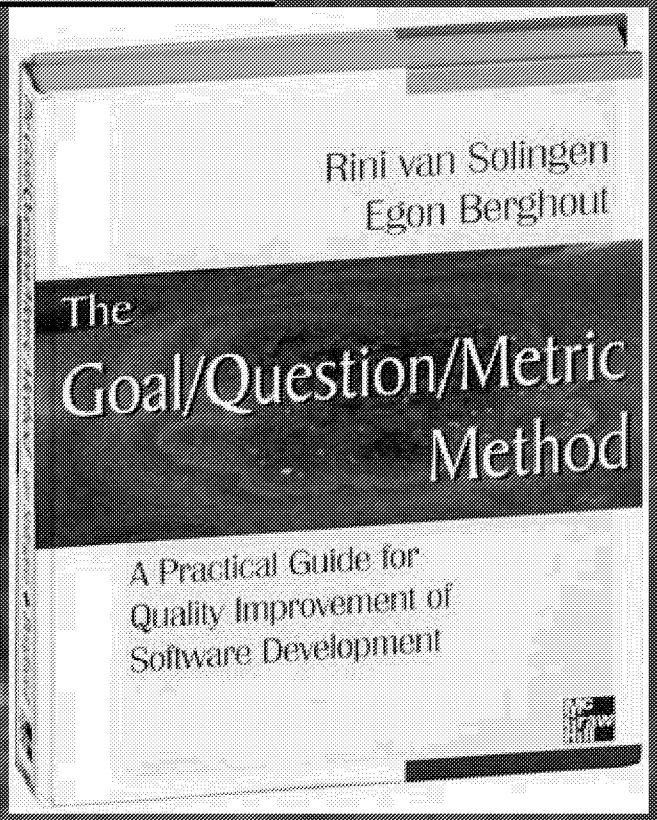
Conclusions

- Positive industrial experiences
- GQM helps focussing
- Specify expectations explicitly
- Project team interprets
- Measurement is also an investment
- Keep it simple
- Combine GQM with SPI efforts

Further reading

- ISBN 0-07-709553-7
- 200 pages, hard cover
- UK£ 35 (= US$ 58)
- SEW'99: US$ 45

  Http://www.gqm.nl/



- IEEE Software Jan/Feb 1998, Reliability
- IEEE Software Sep/Oct 1998, Interrupts

E X P E R I M E N T A T I O N:

Engine for Applied Research and Technology Transfer in Software Engineering

Dieter Rombach

University of Kaiserslautern
Computer Science Department
Software Engineering Chair
Kaiserslautern, Germany
&
Fraunhofer Institute for Experimental Software Engineering (IESE)
Kaiserslautern, Germany

Abstract: The empirical work in NASA's Software Engineering Laboratory in the 70's and 80's has contributed significantly to the maturing of the sub-discipline of 'experimental software engineering'. The development of experimental technologies ranging from the GQM approach for measurement to the EF approach for organizational learning provided the scientific basis; the successful experiments within the SEL development environment served as successful reference examples for others. The Fraunhofer Institute for Experimental Software Engineering (IESE) was founded in Germany based on the successful SEL principles. It was charged with speeding up the transfer of innovative software engineering technologies into a wide variety of industry sectors. The concepts of experimentation were developed further and used for a wide range of purposes from applied research to technology transfer and training. Already during the short history of IESE a successful track record of transferring innovative technologies fast and with sustained success has been established. This presentation focuses on the adaptation of the successful SEL concepts to a different environment, surveys the wide range of applications of 'experiments' as engine for successful technology transfer in a human-based development environment, and predicts a growing importance of experimental work in the future.

1. **Motivation.** The software domain can be characterized by two major facts: (1) The gap between state-of-the-art as taught at universities and state-of-the-practice as 'lived' in most commercial software development environments is_significantly higher than in other

engineering domains, and (2) the body of knowledge available to practitioners consists predominantly of technologies (e.g., languages, techniques, and tools), rather than methods and knowledge regarding the effects of such technologies in practical development environments. One conclusion is that progress in practice is not hindered by lack of technology, but by lack of such latter knowledge which hinders the transfer into practice. Let's just illustrate the problem for one example technology: There exists a very large number of testing techniques today. However, little knowledge exists as to the relative strengths and weaknesses of these techniques in different industrial settings. So, why would a project manager decide to use an alternative testing technique as opposed to the one in use for several years. What is needed can be compared best to so-called 'engineering handbooks' in other engineering disciplines. Such handbooks describe the available technologies together with their applicability, strengths and weaknesses for different constraints. This paper describes how such knowledge can be accumulated in a human-based development environment via experimentation.

2. **Experimentation.** There exist many different ways of accumulating software development knowledge. One very important form of such knowledge is experience derived from actual application of technologies. That means experience is based on product/process feedback loops in that process technology is applied, the impact on the resulting products is observed, and possible improvements regarding the process technology are identified via root cause analysis. In the context of this paper, experience resulting from projects accidentally or experience existing implicitly only is not considered. However, all experiences resulting from systematic hypothesis testing in either fully controlled laboratory experiments or semi-controlled field experiments and field case studies, and producing explicitly sharable insights (models) are considered. Experiments are one of the pre-requisites for sustained learning; it is much easier to change behavior based on documented first-hand experience, rather than knowledge from the world-at-large. Experiments are applicable to basic research for the purpose of understanding, to applied research for the purpose of packaging technologies together with information about their effects in varying project contexts, to teaching & training in order to experience the benefits of new technologies for one's own development tasks before project pressure could result in falling back

to the old technologies for the fear of risk regarding one's own performance, and to technology transfer for the purpose of adapting new technologies optimally to one's project context and providing cost/benefit.

3. **The Role of Experimentation in Software Engineering.** The software domain is characterized by a number of specific characteristics. The most important ones are that most development technologies are human-based and that the data are less frequent and mostly of non-parametric nature. The human-based nature of most technologies makes (a) the change process particularly hard as the 'execution engine' human being needs to be convinced of the benefits of changing to a new technology, and (b) the success of any new technology depends on the adherence to the process guidelines associated with that new technology. Both involves weighing the risk of using the new technology versus the risk of staying with the old technology. Basically, the cardinal question is 'Does it work for ME?'. Experience data from one's very environment are an important source of confidence for changing to and staying with a new technology. The less frequent and mostly non-parametric nature of software engineering data requires different techniques for data analysis – especially the combination of qualitative and quantitative analysis. Beyond that, many of the experimental techniques known from other areas can be applied.

4. **Available Tool Box for Experimental Software Engineering.** The existent body of technologies for experimentation in software engineering itself is significant and growing constantly. Most of the techniques have been initially created in (or have been at least stimulated by) NASA's Software Engineering Laboratory (SEL). Among the most important technologies are

   - the Goal/Question/Metric (GQM) approach for measurement (e.g., [Bas93.1], [Rom91]), supporting the derivation of metrics from a comprehensive goal specification
   - the Quality Improvement Paradigm (QIP) method (e.g., [Bas93.2]), enabling the integration of sound project feedback for project control with cross-project learning (NOTE: It adapts the Plan/Do/Check/Act approach from manufacturing to the specifics of the software domain)

- the Experience Factory (EF) approach (e.g., [Bas93.2]), defining extra learning related roles and integrating them with the traditional software development roles
- a portfolio of experimental designs (e.g., [Bas86]), ranging from controlled experiments to regular field case studies
- a variety of analysis methods (e.g., [Bri92]) for non-parametric software engineering data, integrating qualitative and quantitative analysis techniques

In addition, there exist

- a number of reference laboratory environments applying the above experimental technologies such as NASA's SEL as the 'mother of all laboratory environments', Fraunhofer IESE in Germany, and CAESAR in Australia
- a number of exchange forums such as the International Network for Software Engineering Research (ISERN) for researchers or the Software Experience Consortium (SEC) for practitioners
- a growing number of conferences (e.g., METRICS , SEL Workshop) and journals (e.g., International Journal for Empirical Software Engineering)

All this provides a sound starting point for experimental work. The ISERN Network is open to everybody interested in further developing the experimental technologies, teaming up in concrete technology experiment replication, and exchanging all kinds of experiences. The contact address is 'isern@informatik.uni-kl.de'. The SEC Consortium is open for application by companies active in the area of empirical work or corporate experience management. The contact address is 'fshull@fc-md.umd.edu'.

5. **Fraunhofer IESE: An Institute built on the Experimental Paradigm.** The Fraunhofer Gesllschaft e.V. in Germany is Europe's largest applied research and technology transfer organization. It consists of 48 institutes ranging in application domain from material sciences and production technology to information & communication technology and life sciences. These institutes receive approximately 30% base funding from government; the remaining 70% of their operating budgets have to be covered from industry project income.

The Fraunhofer Institute for Experimental Software Engineering (IESE) became the 48th permanent Fraunhofer Institute [Rom96]. Founded in 1996, its area of competence is software engineering; its applied research and transfer model is based on the experimental paradigm. That means Fraunhofer IESE helps companies to establish experimentally based learning organizations as a pre-requisite for sustained improvements, and then helps them introduce new innovative software development technologies (technical & managerial). With the base funding from government, technologies from basic research institutions are being evaluated via experimentation, and packaged together with the experimental results for transfer into specific domain and company environments.

Today, Fraunhofer IESE employees 80 full time scientists together with about 60 part-time personnel such as students and consultants. Th institute language is 'English'; 25% of personnel is non-German. The percentage of industrial income has risen to about 70% within three years. Collaborations include a large number of Europe's leading companies in the sectors of telecom, automotive & aerospace, and banking/insurance/trade.

Fraunhofer IESE has been created as the German instantiation of the NASA/SEL laboratory model. It was widely accepted that a closer collaboration between academia and industry was needed. This institutionalized model – allowing for long-term trusting relationships between academia and companies – was the answer. The reference to the working SEL example was one of the major arguments to finally convince companies and government of the opportunity at hand. Many of the concepts of IESE are based on SEL experiences by myself during my tenure at the University of Maryland and my involvement with NASA/SEL during the 1986-1991 time frame.

The main SEL concepts adopted include

- provision of an environment in which researchers, software developers, and customers can work together
- use of experimentation as a major research and technology transfer engine
- establishment of long-term relationships with development organizations

- exposing researchers to practice and practitioners to research
- have research being driven by practical needs (= applied research!)

However, there are some important differences compared to the SEL. They include

- operation as a business due to the fact that Fraunhofer Gesellschaft e.V. is a legal non-for-profit entity not associated with any university or for-profit company environment (business plan for 140 employees!)
- tougher sales job for close academia/industry collaboration due to a historically wider gap between academia and industry in Germany as compared to the US
- need for critical mass in IESE core competence areas personnel-wise due to the expectation by companies to support them strategically (i.e., long-term, always with experienced personnel)
- need for application sector know-how in addition to software engineering competence due to the fact that IESE collaborates with companies from different industry sectors
- need for complex incentive structure in order to provide equal motivation to researchers and practitioners working in IESE

Although, many of the experiences and lessons learned within the SEL could be reused, the changes due to the collaboration culture and heterogeneity in customer base posed the biggest challenges. However, the achieved high standing of IESE within the scientific and industrial community demonstrates the possibility of replicating the SEL experience.

6. Useful Applications. This section describes briefly some of the typical applications of the experimental paradigm within the Fraunhofer IESE. These applications comprise – due IESE's mission – applied research, teaching & training, and technology transfer. It is intended to describe the wide applicability and usefulness of experimentation – even in a very industry oriented setting.

6.1. Applied Research. It has been firmly established at IESE that applied research in software engineering produces new/refined/existing technologies together with recorded observations regarding their effectiveness in one or a class of industrial setting (i.e., certain

constraints). These observations need to be produced by some appropriate form of experimentation. These observations are only useful, if the underlying experiment is documented well enough to be repeatable by anyone challenging the findings or trying to replicate them in a slightly different environment. Observations from non-repeatable experiments do not contribute to the state-of-the-art. In that context, it must also be agreed that experiments with negative results are equally valuable. Negative results combined with qualitative analysis investigating possible causes and deriving new hypotheses contribute to learning. There exist only badly designed and/or performed experiments, no bad results!

Such experiments have been done for most of the IESE technologies ranging from software development to management and experimental technologies. The most prominent experiments include the

- effectiveness & efficiency of step-wise abstraction code reading (e.g., [Bas87])
- effectiveness & efficiency of perspective-based requirements reading (e.g., [Bas96])
- maintainability of well-structured OO programs (e.g., [Bri97])
- maintainability of well-documented (traceability from requiremens to code) programs
- cost/benefit ratio for product line development

All these experimental results are published in the literature. Most of them are accessible through the IESE web site. More experiments on the above as well as other topics are needed. Every software engineering researcher should feel challenged to participate. The International Software Engineering Research Network (ISERN) provides a stimulating environment to learn, share and collaborate. Please contact ISERN (www.iese.fhg.de/ISERN/, isern@informatik.uni-kl.de)!

6.2. **Teaching & Training.** Software engineering teaching and training must include the topic of experimental methods (see e.g., CMSC735 at the University of Maryland OR SE2 at the University of Kaiserslautern) as well as their practical application to self-experience important software engineering principles (see examples from the University of Kaiserslautern below!). The simple lecturing of software

engineering principles results too often in them being ignored during the next development tasks. Again the issue is that changing behavior requires motivation that the risk of change is manageable. Experiments as part of teaching can provide the necessary motivation. During practical industrial training such experiments can be repeated for the same reason of motivation for change. In addition, experimentation can demonstrate the applicability of some technology to the specific company setting and suggest some necessary adjustments prior to real use.

Together with the University of Kaiserslauternn Fraunhofer IESE has developed a number of technology demonstration experiments which are being repeated during every graduate level software engineering class as well as during industrial training (modified according to company constraints!). The standard experiments include

- demonstrating the superiority (i.e., effectiveness, efficiency) of code reading over unit testing (adaptation of the old 'Selby' experiment) (e.g., [Lot96])
- demonstrating the superiority (better understandability, modifiability) of well-structured OO designs over worse structured ones
- demonstrating the superiority (better modifiability) of tractably documented programs over less tractably documented ones
- demonstrating the superiority (i.e., effectiveness, efficiency) of perspective-based reading of informal requirements over other reading techniques

Each of these experiments has been performed at least three times. Comprehensive laboratory packages are available describing the experiment and providing key artifacts for easy replication in other environments.

6.3. <u>Technology Transfer.</u> The purpose of experimentation in technology transfer is twofold: First before the introduction of a candidate new technology experimentation helps to convince personnel (top management to invest in it, project management to support it, and project personnel to 'live' it under project pressure) of the potential benefits of a pre-packaged new technology, and it helps to adapt pre-packaged technology to specific needs of the target organization. Second during use of the new technology

experimentation helps to change the technology further in order to optimize its effects, and it helps to re-enforce its continued use and, as a result thereof, ensures its continued gains.

During its 3 year history Fraunhofer IESE has contributed to many sustained process improvements in industry which would have been impossible without experimentation (e.g., [Lai97]). An extensive list of company references can be obtained from the IESE web site.

7. **Outlook.** Experimentation is becoming an integral sub-discipline of software engineering. Reflecting the general needs of an engineering discipline and the specific characteristics of the software domain, a body of technologies and reference applications have been created. The role of NASA/SEL has been equally instrumental to the area of experimentation as has been the SEI's role to the area of assessments. NASA/SEL together with its off-springs (e.g., Fraunhofer IESE) has pioneered the application of experimentation to speed up the accumulation of shareable, testable & repeatable knowledge in research, to raise a generation of true software engineers thru teaching and training, and to speed up the infusion of innovative software development technologies into practice in technology transfer programs. More and more environments will recognize that experimentation does not represent additional effort, but rather speeds up the production of real contributions to the state-of-the-art in software engineering and their transfer into practice. As the performance of real experiments require laboratory set-ups at universities or in companies, more of such environments must be established.

   I wish the SEL a successful future! May it spin off more laboratory environments around the globe! May it be valued inside NASA as highly as it is outside!

8. **References.**

→ [Bas93.1] Basili, Caldiera & Rombach, The GQM Paradigm, in 'Encyclopedia of Software Engineering' (John J. Marciniak, Ed-in-Chief), John Wiley & Sons, Inc., 1993.

→ [Bas93.2] Basili, Caldiera & Rombach, The Experience Factory, in 'Encyclopedia of Software Engineering' (John J. Marciniak, Ed-in-Chief), John Wiley & Sons, Inc., 1993.

→ [Bas87] Basili & Selby, Comparing the Effectiveness of Software Testing Strategies, IEEE Transactions on Software Engineering, vol. 13, no. 12, pp. 1278-1296, December 1987.
→ [Bas86] Basili, Selby & Hutchens, Experimentation in Software Engineering, IEEE Transactions on Software Engineering, vol. 12, no. 7, pp. 733-743, July 1986.
→ [Bas96] Basili, Laitenberger, Shull et al, The Empirical Investigation of Perspective-Basded Reading, International Journal of Empirical Software Engineering, vol. 1, no. 2, pp. 133-164, 1996.
→ [Bri92] Briand, Basili & Thomas, A Pattern Recognition Approach to Software Engineering Data Analysis, IEEE Transactions on Software Engineering, vol. 18, no. 11, pp. 931-942, November 1992.
→ [Bri97] Briand, Bunse et al, An Experimental Comparison of the Maintainability of Object Oriented and Structured Design Documents, International Journal of Empirical Software Engineering, vol. 2, no. 3, pp. 291-312, 1997.
→ [Lai97] Laitenberger & DeBaud, Perspective-Based Reading of Code Documents at Robert-Bosch GmbH, Information & Software Technology, vol. 39, pp. 781-791, 1997.
→ [Lot96] Lott, Rombach, Repeatable Software Engineering Experiments for Comparing Defect-Detection Techniques, International Journal of Empirical Software Engineering, vol. 1, no. 3, pp. 241-277, 1996.
→ [Rom91] Rombach, Practical Benefits of Goal-Oriented Measurement, in Software Reliability and Metrics (Fenton & Littlewood, Eds.), Elsevier Publ., 1991.
→ [Rom96] Rombach et al, New Institute for Applied Software Engineering Research, International Software Process Journal, vol. 2, no. 2, 1996.

9. **Contacts**. For further information about this paper, please contact the author under 'rombach@iese.fhg.de'. For further information regarding the Software Engineering Chair at the University of Kaiserslautern, please check 'wwwagse.informatik.uni-kl.de'; for further information about the Fraunhofer Institute IESE, please check 'www.iese.fhg.de'. For information about the International Software Engineering Research Network (ISERN), please check 'www.iese.fhg.de/ISERN/' or contact 'info@iese.fhg.de'. For information about SEC Consortium, please contact 'fshull@fc-md.umd.edu'.

EXPERIMENTATION:
Engine for Applied Research and Technology Transfer in Software Engineering

Dieter Rombach

University of Kaiserslautern
Computer Science Department
Software Engineering Chair
Kaiserslautern, Germany
wwwagse.informatik.uni-kl.de

Fraunhofer Institute
for Experimental Software
Engineering (IESE)
Kaiserslautern,Germany
www.iese.fhg.de

UNIVERSITÄT KAISERSLAUTERN
AG·SE INSTITUT
FACHBEREICH INFORMATIK

IESE
Fraunhofer Einrichtung Experimentelles Software Engineering

Motivation

- Gap between 'state of research' and 'state of practice' in software engineering much wider than in other technology fields!
- Many innovative technologies not lived in practice
- Body of knowledge in software engineering dominated by
  - languages, techniques & tools
  - rather than methods and knowledge regarding their effects in real settings

UNIVERSITÄT KAISERSLAUTERN
AG SE
FACHBEREICH INFORMATIK

Fraunhofer IESE
Einrichtung Experimentelles Software Engineering

Experimentation:

- Experience is knowledge based on actual application
- Experimentation comprises all forms of systematic hypothesis testing leading to explicitly documented and therefore sharable experiences
- Experimental forms range
  - from fully controlled laboratory experiments
  - to case studies
- Such experimentation is
  - a prerequisite for sustained learning & improvement
  - applicable in research/teaching&training/tech transfer

UNIVERSITÄT KAISERSLAUTERN
AG·SE INSTITUT
FACHBEREICH INFORMATIK

Fraunhofer IESE
Einrichtung Experimentelles Software Engineering

Role of Experimentation in Software Engineering:

- Most software engineering technologies are human-based
- Experimentation enables sustained learning
  - about the effects of new technologies (e.g., effort)
  - about the impact of (human) variation factors (e.g., experience with application)
  - 'whether it works for ME?' (motivation for process conformance)

UNIVERSITÄT KAISERSLAUTERN
AG·SE INSTITUT
FACHBEREICH INFORMATIK

Fraunhofer IESE
Einrichtung Experimentelles Software Engineering

Available "Tool Box" for Experimentation in SE:

- GQM measurement approach (Basili et al)
- QIP experimental method (Basili et al)
- EF laboratory set-up (Basili et al)
- Experimental design types (Selby et al)
- Analysis methods (for non-parametric SE data) (Briand et al)
- Reference laboratories (e.g., SEL, IESE, CAESAR)
- Exchange forums (e.g., ISERN, SEC)
- Conferences & journals (e.g., Metrics, SEL-WS, JESE)

UNIVERSITÄT KAISERSLAUTERN
AG·SE
FACHBEREICH INFORMATIK

Fraunhofer IESE
Einrichtung Experimentelles Software Engineering

Fraunhofer IESE: ... built on the Experim. Paradigm:

- Institute for Experimental Software Engineering (IESE)
  - experimental process engineering
  - enables predictable product engineering
- Founded on 01 January 1996
- Mission
  - help German/European/... companies in
    - building up learning organizations for software business based on QIP/EF approach
    - transferring innovative software development technologies into practice fast and with limited risk

UNIVERSITÄT KAISERSLAUTERN
AG·SE
FACHBEREICH INFORMATIK

Fraunhofer IESE
Einrichtung Experimentelles Software Engineering

Fraunhofer IESE: ... built on the Experim. Paradigm:

- Inheritance from NASA's SEL
  - Concept of software engineering laboratory environment was adapted to Germany
    - customer/developer/researcher collaboration
    - experimentation as major research/transfer vehicle
    - long-standing relationship to build up mutual trust
  - I was trained/learned in SEL environment (84-91)
    - dual role of university research & practice involvement
    - research & technology development focused on practitioner's needs (= applied research)
  - Reference to SE was key argument to get govern-ment & industry support in Germany (in 1995!)

UNIVERSITÄT KAISERSLAUTERN
AG·SE INSTITUT
FACHBEREICH INFORMATIK

Fraunhofer IESE
Einrichtung Experimentelles Software Engineering

Fraunhofer IESE: ... built on the Experim. Paradigm:

- **Differences compared to NASA's SEL**
  - separate legal entity

    Provide business plan for 140 employees!
  - different culture regarding industry/university collaboration

    Convince academics & industry of benefits!
  - strategic relationships with many companies from many different sectors (IT, Telecom, Automotive, Aerospace, Banking/Insurance)

    Provide critical mass in core competence areas!
    Combine application knowledge with SE competence!
  - collaboration with competing companies

    Provide solid mechanisms for confidentiality/security!

UNIVERSITÄT KAISERSLAUTERN
AG·SE
FACHBEREICH INFORMATIK

Fraunhofer IESE
Einrichtung Experimentelles Software Engineering

Useful Applications ... (Applied Research):

- Applied software engineering research should produce
  - new/refined technologies
  - together with experiences regarding its effects
  - in a specific setting (constraints)
- Experiences have to be the result of some form of experimentation
- Experiments have to be repeatable
  - requires sound documentation (in publications)
  - otherwise no contribution to state-of-the art
- There are no unsuccessful, only poorly designed experiments
  - rejections of hypotheses are also valuable
  - qualitative analysis points out reasons for rejection = new hypotheses!

UNIVERSITÄT KAISERSLAUTERN
AG·SE INSTITUT
FACHBEREICH INFORMATIK

IESE
Fraunhofer Einrichtung Experimentelles Software Engineering

Useful Applications ... (Applied Research):

- Examples
  - step-wise abstraction reading of code is more effective & efficient for defect detection than testing and other reading techniques
    - initial study in SEL [Selby et al]
    - replications at universities (e.g., Kaiserslautern [Lott])
    - replications in companies (e.g., Bosch Telecom)
  - perspective-based reading of informal requirements is effective and efficient for defect detection than other reading techniques
    - initial study in SEL [Basili et al]
    - replications (e.g., Kaiserslautern [Laitenberger])

UNIVERSITÄT KAISERSLAUTERN
AG SE
FACHBEREICH INFORMATIK

Fraunhofer IESE
Einrichtung Experimentelles Software Engineering

Useful Applications ... (Applied Research):

- More studies on more topics are needed!
- Join the International SE Research Network
  - -> www.iese.fhg.de/ISERN/
  - -> isern@informatik.uni-kl.de

UNIVERSITÄT KAISERSLAUTERN
AG SE
FACHBEREICH INFORMATIK

Fraunhofer IESE
Einrichtung Experimentelles Software Engineering

Useful Applications ... (Teaching & Training):

- SE education should include
  - teaching of experimental methods
  - their application to 'self experience' important sw engineering principles (e.g., inf. hiding)
- SE education without experimentation will not affect long-term development behavior
- SE training should include
  - 'experimentation' with new technologies to judge their applicability in work context

UNIVERSITÄT KAISERSLAUTERN
AG·SE INSTITUT
FACHBEREICH INFORMATIK

Fraunhofer IESE
Einrichtung Experimentelles Software Engineering

Useful Applications ... (Teaching & Training):

- Experimentation is class subject (e.g., CMSC 735)
- Examples of graduate software engineering class experiments in Kaiserslautern
  - experience the superiority of code reading over unit testing
  - experience the advantages of good OO designs (inf. Hiding, etc.) for maintenance
  - experience the benefits of tractable documentation for maintenance
  - experience the superiority of perspective-based reading of informal requirements over other reading techniques

UNIVERSITÄT KAISERSLAUTERN
AG·SE INSTITUT
FACHBEREICH INFORMATIK

Fraunhofer IESE
Einrichtung Experimentelles Software Engineering

Useful Applications ... (Teaching & Training):

Lab manuals of all classroom experiments are available for replication!

UNIVERSITÄT KAISERSLAUTERN
AG·SE INSTITUT
FACHBEREICH INFORMATIK

Fraunhofer IESE
Einrichtung Experimentelles Software Engineering

Useful Applications ... (Technology Transfer):

- Technology transfer
  - convinces personnel of benefits
    - top management to invest
    - project management to support
    - project personnel to 'live' new technology
  - adapts generic technologies to company needs
  - optimizes new technologies within organization
  - re-enforces use and sustains gains

UNIVERSITÄT KAISERSLAUTERN
AG SE INSTITUT
FACHBEREICH INFORMATIK

Fraunhofer IESE
Einrichtung Experimentelles Software Engineering

Useful Applications ... (Technology Transfer):

- Example (from Fraunhofer IESE)
  - Bosch Telecom, Germany (private networks)
  - Problems with late problems (and rework)
  - Idea: Introduce 'PBR-based inspections for requirements'
  - Procedure (3 years, about 6 PY effort on IESE side)
    - Step1: Reference existing experiments (SEL, IESE)
    - Step 2: Repeat PBR experiment with developers
    - Step 3: Adapt/package PBR for use in pilot project
    - Step 4: Evaluate results from pilot project (- 40% rework!)
    - Step 5: Optimize effects in follow-ups (> -90% rework!)
    - Step 6: Roll-out to all projects (Frankfurt)
    - Step 7: Roll-out to different sites (Germany, France)

UNIVERSITÄT KAISERSLAUTERN
AG·SE INSTITUT
FACHBEREICH INFORMATIK

IESE
Fraunhofer Einrichtung Experimentelles Software Engineering

Useful Applications ... (Technology Transfer):

Companies experience

- **sustained improvements**
- **return on investments from improvements**

UNIVERSITÄT KAISERSLAUTERN
AG·SE
FACHBEREICH INFORMATIK

Fraunhofer IESE
Einrichtung Experimentelles Software Engineering

Outlook:

- Experimentation will become an essential sub-discipline of software engineering
  - to speed up the accumulation of shareable, testable & repeatable knowledge (research)
  - to raise a generation of true software engineers (teaching & training)
  - to speed up the infusion of innovative technologies into practice (technology transfer)

Experimentation is the engine to speed up the transfer of applied research results into practice!

UNIVERSITÄT KAISERSLAUTERN
AG·SE
FACHBEREICH INFORMATIK

Fraunhofer IESE
Einrichtung Experimentelles Software Engineering

Outlook:

- More organizations (academic laboratories & industrial learning organizations) need to be based upon the experimental paradigm
- The SEL has been for ESE what the SEI has been for assessment! It started a movement!
  - Fraunhofer IESE is an offspring of NASA's SEL

I wish the SEL a successful future!
May it be valued inside NASA as highly as it is regarded outside!

UNIVERSITÄT KAISERSLAUTERN
AG·SE
FACHBEREICH INFORMATIK

IESE
Fraunhofer Einrichtung Experimentelles Software Engineering

Software Experience Center: The Evolution of the Experience Factory Concept

Frank Houdek and Kurt Schneider
DaimlerChrysler AG
Research and Technology
P.O. Box 23 60
D-89013 Ulm, Germany
{frank.houdek,kurt.schneider}@daimlerchrysler.com

Abstract

The experience factory concept, which was evolved at the NASA Software Engineering Laboratory, is a promising concept geared at facing the current needs in software development and software process improvement. Therefore, we at DaimlerChrysler decided to implement it in several business units to maintain and improve software engineering competence. In our efforts to establish the experience factory concept, we identified some shortcomings resulting from (unstated) assumptions. In this paper, we point out these assumptions and present how we evolved the experience factory concept. In particular, we introduced reinfusion concepts, concepts for experience evolution and for cost/benefit-ratio of experience items. An example taken from our business units helps to concretize our findings.

1 Introduction

Software engineering knowledge is becoming more and more a strategic business competence — both for software and system developing companies. The ability to produce high-quality software within a reasonable time and budget is becoming critical for economic success.

The *experience factory concept* developed by Basili and co-workers in a collaboration with the NASA Software Engineering Laboratory, the University of Maryland and the Computer Science Corporation [Bas89, BCM+92, Bas93, BCR94, BC95, BM96] is a promising approach to build up and maintain software engineering knowledge related to the specific needs of an enterprise.

Hence, in 1997 DaimlerChrysler decided to implement the experience factory concept within several software-intensive business units [HSW98, LSH99, WHS99, HB99]. In particular, we started initiatives in passenger car development, military aircraft development and central IT services, each of which is by the corporate research department. Our mission was to establish the experience factory concept within two to three years. The overall

goal of the initiatives was to improve software development competencies. The individual goals, however, varied in the business units depending on their specific demands. Central IT services, for instance, was interested in improving their software contracting processes, whereas, in passenger car development, defect profiles and defect tracking were important concerns.

At the beginning of our initiatives, we tried to instantiate the SEL's experience factory concept by building up an independent organizational unit, defining experience documentation procedures and running measurement programs. But these activities are long-term activities and the business units involved were also seeking short-term benefits. Their motivation to act as partners in the experience factory initiatives was to achieve significant improvements in their situation and sustain it within the initiative schedule. As a consequence, we were forced to evolve the experience factory concept in order to initiate short- and long-term experience-based improvements in parallel. We call the resulting approach *software experience center* (SEC) and we will discuss our findings in this paper in some detail.

1.1 Structure of the Paper

Section 2 briefly summarizes the SEL experience factory concept and emphasizes the assumptions behind it. Building on this concept, we introduce three necessary dimensions of evolution for the 'classic' experience factory concept in Section 3. Section 4 gives an example illustrating the evolved concepts. Section 5 summarizes the findings of these paper.

2 The Origin: SEL Experience Factory

Process improvement is hard work. Deficits have to be identified, improvement activities must be defined and implemented, and their effects monitored. This is how most improvement approaches work. However, these activities are only partially useful for a single project which has to create a product within a given schedule and cost frame. To make improvement activities successful in the long run, projects concerns have to be clearly separated from improvement concerns.

This insight was the main trigger for the experience factory concept, which is based on the *quality improvement paradigm* (QIP, see [BCR94]). The experience factory concept proposes a (logical and organizational) separation of project organization (responsible for building products) and improvement organization (responsible for improving processes within and across projects). The experience factory organization supports individual projects by providing them with experience gained from work in previous projects. The observations made in the new project are, in turn, used to update the organization's experience base (see Figure 1). And, a cross-project learning process becomes alive.

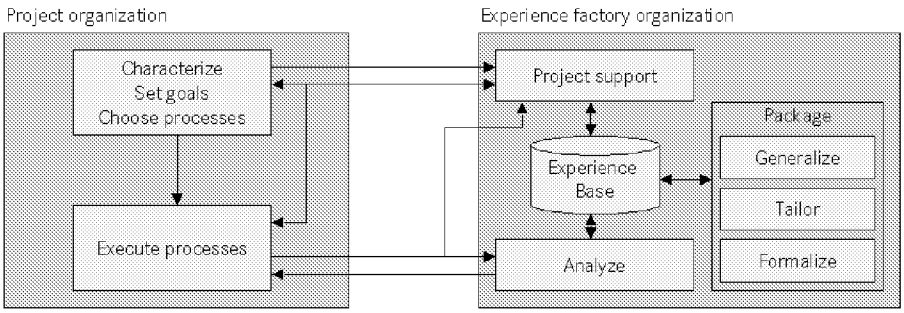


Figure 1: Experience factory concept [BCR94].

This concept is rather obvious. It helped to clear our mind. In many improvement projects at DaimlerChrysler it helped us a lot [HSW98]. The experience factory concept provides a long-term vision for improvement initiatives tailored to particular business units needs.

Even though this concept is obvious, it has several implications and makes several assumptions:

- *Long-term activity.* The improvement approach underlying the experience factory concept is the QIP. According to the QIP, process improvements are, by their nature, long-term. First, the actual situation has to be basedlined. Then, improvement activities are defined, implemented and assessed. Typical time-frames for QIP-based improvements are one to three years.

- *Additional effort.* From the perspective of a single project, process improvement and learning require additional resources (e.g. for measuring) which do not pay off immediately.

- *Common understanding.* An important step in every improvement initiative is defining an improvement goal. To do so, people need to know and articulate their needs accordingly.

- *Similar projects.* The basic idea of the experience factory concept is to learn in one project and to transfer the gained experience to another one. The essential prerequisite is that both projects are sufficiently similar.

- *Processes in place.* Process improvement requires fairly mature processes that are beyond the ad-hoc stage.

- *'Homo economicus'.* Improvement activities have similarities with farming. One has to seed now (spending some effort) to harvest (some more) in the future. Common sense tells us that this is a reasonable

thing to do. However, humans do not always act reasonably with respect to long-term economic considerations.

- *Will to change.* Improvement is almost always tied with changes: changing processes, changing responsibilities, changing personal behavior. But changing is never easy. Although it is reasonable to change, people are often reluctant to do so.

- *Pull for external knowledge.* Learning across projects is essential in the experience factory concept. This means, that people are willing to learn and willing to accept knowledge and experience gained in other environments (i.e., projects). Moreover, people have to ask for knowledge, trawl for experience items, seek for better processes. So there must be an active *pull* for helpful information.

- *Management support.* Every change needs a powerful sponsor. To bring the experience factory concept to life, permanent support from powerful sponsors (i.e. management at all levels) is mandatory.

In most environments, there are some deficits concerning the issues mentioned above. In particular, a long-term commitment at all levels (management, project members) is hard to uphold. An external observer would argue that it is worthwhile to spend effort for activities whose return on investment is not immediately yielded. However, project workers who are permanently 'up to their necks in hot water' have a slightly different perception. They can accept only short-term initiatives. They want to see improvement right now.

For these reasons it is not sufficient to introduce the 'classic' experience factory concept in a 'typical' organization. In the next section, we show how we have evolved the experience factory concept to cope with the above mentioned issues.

3 Dimensions of Evolution

In our experience factory initiative at DaimlerChrysler, we began with the 'classic' experience factory concept. But after a short time it became obvious that it is impossible to uphold the long-term commitment required without short-term benefits for the persons involved (see [HSW98, WHS99]). We were forced to evolve the 'classic' experience factory concept.

Figure 2 sketches the identified dimensions of evolution graphically. In the following, we focus on them in some detail:

- *Reinfusion concepts.* The 'classic' experience factory concept emphasizes experience collection: for example, measurement programs,

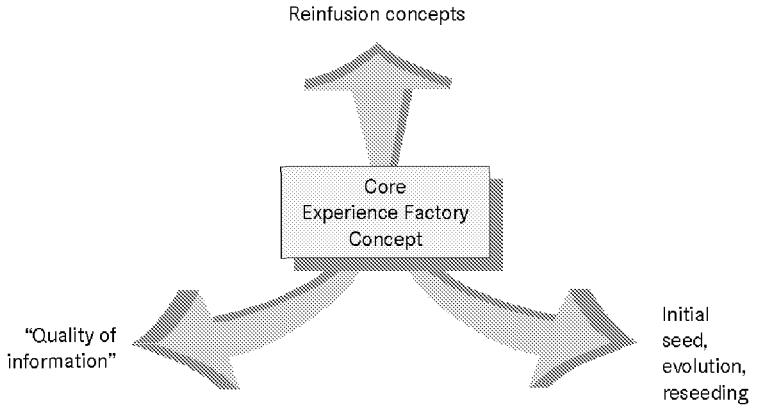


Figure 2: Dimensions of evolution.

model building, formalization and generalization. Reinfusion of experience, i.e. delivery of experience items into projects, is seen to happen naturally (after some tailoring).

This assumption does not hold for several reasons: (1) People do not ask for relevant experience items by themselves. Typically, they do their job as best they can. Therefore, it is crucial to provide experience items for the task at hand at the right time [FLO+96, LS97]. (2) People do not know that they might need some additional experience items. Either they assume there is nothing relevant in the experience base or they do not even recognize their current job as being experience-intensive.

- *'Quality of information.'* Measurement based-information and derived models are the prime experience items provided by the 'classic' experience factory. This type of experience is desirable because it provides detailed and objective information. However, gathering it is laborious and time-consuming. The time delay from experience collection to harvested benefit is fairly long (sometimes several years). Staying alive in view of short-time expectations, experience items with shorter reuse-cycles are also required. Of course, their potential benefit might be only slight, as the information is less consolidated and more subjective. Figure 3 gives examples of different types of experience items. It also qualitatively depicts the trade-off between the effort needed to build a particular experience item and its expected benefit. Before building a new experience item, the utility (i.e. the ratio of expected benefit and needed effort) should be assessed.

- *Initial seed, evolution and reseeding.*

  The 'classic' experience factory concept is driven by the QIP. This

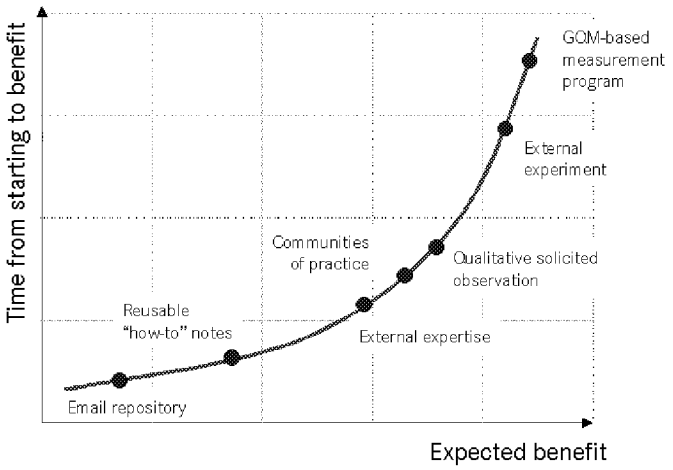


Figure 3: Quality of information.

implies that there is always a clear goal for improvement activities and experience collection. This assumption does not hold in practice. We often observe a moving-target situation, i.e. the goal and the associated needs change over time, sometimes by accident, sometimes due to the experience items delivered. A more dynamic approach is, thus, needed to avoid wasting a great deal of effort for experience-building activities (e.g. measurement programs) which provide experience items that are not really helpful. A closed feedback-loop of seeding (i.e., providing some cheap experience items), evolution (of needs) and reseeding (i.e. adjusting experience items and experience collection processes) is essential [Fis98].

4 Example

In this section, we present an example taken from our experience factory initiatives to illustrate how we evolved the 'classic' experience factory concept in practice.

This example is from the central IT services business unit. This unit is involved in large projects developing systems for administrative purposes like global sales, warranty management or diagnosis. Typically, such systems are not built in-house but contracted out to one or more suppliers. Central IT services is responsible for contractor management and associated activities such as acceptance processes or quality definition.

Our mission (corporate research) was to establish an initial experience factory group there. The experience items they were to maintain were aimed at supplementing all the activities concerned with contracting software out and performing acceptance tests at delivery time. In the beginning of our

activities, we acted as 'experience factory guys'. With time, people from the central IT services were to take over our roles.

We started (according to the QIP) with an extensive baselining to identify the existing processes, quality needs, etc.

During this work, we encountered a 'pull' situation, i.e. demand for experience items (for, in this case, processes for contract evaluation) arose. This issue has not been covered by the currently implemented experience factory activities so far. Setting up a serious analysis of existing processes (as the 'classic' experience factory concept would imply) would have resulted in a long-term activity. Instead, we performed interviews, studied relevant literature and (company) standards, tailored the findings towards the actual needs and provided simple guidelines (how-to notes).

Founding on our baselining activities, we encountered some other questions which were not directly articulated by the projects but which might become vital in future activities (e.g. risk assessment, role of quality manager). Consequently, we also built experience items for these topics. Unlike the contract evaluation process item, we had to sell these experience items. This was mainly, because people were not aware of the utility of these issues (e.g. risk management).

We used selling and applying experiences to improve the existing experience items. Figure 4 depicts the flow of experience across several projects in the central IT services business unit. However, there is no indication whether a flow of information was initiated by pull or push. There were variations across projects and over time.

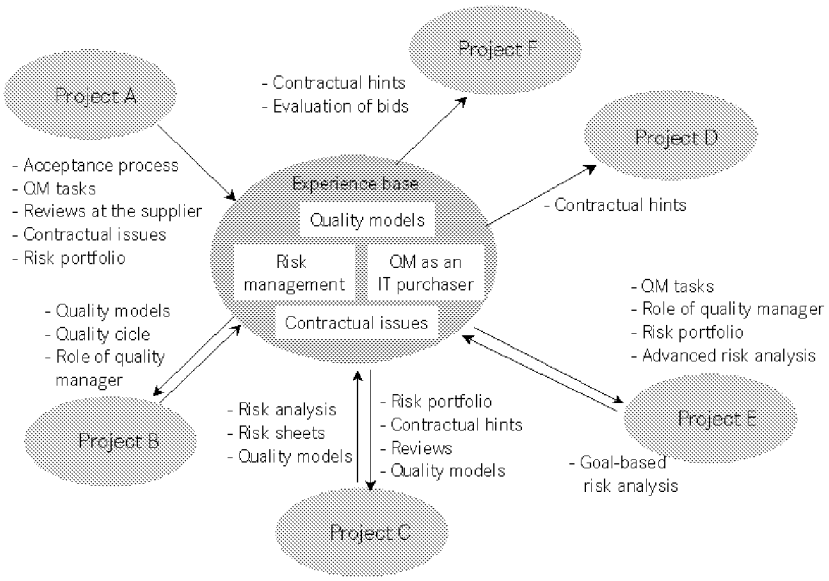


Figure 4: Experience transfer at central IT services.

It is important to realize that there was neither an initial pull for most of these items nor a clear agreement that these and only these items were relevant.

With respect to the above mentioned evolution dimensions, we made the following contributions:

- The created experience items were fairly cheap to build with only a limited benefit but a positive cost/benefit ratio (e.g. checklist for contracts, initial quality model). Experience items of better quality were only built in cases where return on investment was anticipated within a reasonable time-frame. Especially the fact that the experience factory initiative showed benefit to the projects within a rather short time helped us greatly to become accepted in this business unit.

- It was not clear from the beginning where to go as the people involved were unable to articulate their particular needs. So we were not able to start with a clear goal in mind but had to work iteratively and providently. We started with an initial seed (hints on writing contracts) and evolved the experience base content over time. Starting a QIP program would have not produced the same output. The goal identification would only have raised topics which the people were aware of. However, we found some items to be extremely helpful which would not hove been raised as people did not know them.

- Rarely did people seek for experience items. So the assumption that a filled experience base is enough to make an experience factory helpful proved to be false.

  More often, we (as the experience factory guys) had to push our items in meetings and project planing sessions. More details on the relation of pull versus push (which is the main reason for reinfusion concepts) can be found in [WHS99].

5 Summary

The experience factory concept which was evolved at the NASA Software Engineering Laboratory is a promising concept geared at the current needs in software development and software process improvement. It addresses the burning issues of a business unit rather than proposing one-size-fits-all solutions.

To understand its transferability to other environments, it is important to understand its evolution and its assumptions. The experience factory concept is the outcome of many years of work performed by Basili and coworkers [BCM+92] at the NASA SEL. It is a result of process improvement activities according to the PDCA principle (i.e. QIP [BCR94]) and the

perception that successful improvement activities must be separated organizationally from project work. At SEL, it was never the goal to 'build an experience factory', but the resulting organization was *ex-post* called an experience factory after it grew for several years.

If you intend to establish the experience factory concept within a fairly short time-frame (e.g. two to three years), some shortcomings of the concept become obvious resulting from (unstated) assumptions behind the concept. Primarily, it is assumed that project people believe in the (long-term) benefits of an experience factory.

In our experience factory initiative at DaimlerChrysler, we identified some areas in the experience factory concept that need to be evolved. In particular, we recognized the need for reinfusion concepts, concepts for experience evolution and a continuum of experience items ranging from easy-to-build but short-term-benefit items (e.g. how-to notes, expert networks) to solid high-impact packaged experience (e.g., results from GQM measurement programs).

We call the evolved experience factory concept the 'software experience center' (see Figure 5).

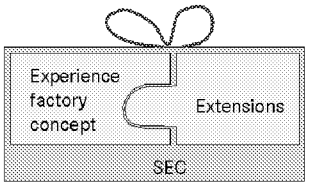


Figure 5: Software experience center.

Acknowledgements

I would like to acknowledge the contribution of everybody involved in our experience factory initiatives. Without their collaboration, patience and encouragement our efforts would not have been possible.

References

[Bas89] V.R. Basili. The experience factory: packaging software experience. In *Proceedings of the 14^{th} Annual Software Engineering Workshop.* NASA Goddard Space Flight Center, Greenbelt MD 20771, 1989.

[Bas93] V.R. Basili. The experience factory and its relationship to other improvement paradigms. In I. Sommerville and M. Paul, editors, *Proceedings of the 4^{th} European Software Engineering Conference (ESEC)*, number 717 in Lecture Notes in Computer Science, pages 68–83. Springer Verlag, Berlin, September 1993.

[BC95] V.R. Basili and G. Caldiera. Improve software quality by reusing knowledge and experience. *Sloan Management Review*, 37(1):55–64, 1995.

[BCM+92] V. Basili, G. Caldiera, F. McGarry, R. Pajerski, and G. Page. The Software Engineering Laboratory — An operational software experience factory. In *Proceedings of the 14th International Conference on Software Engineering (ICSE)*, pages 370–381, May 1992.

[BCR94] V.R. Basili, G. Caldiera, and H.D. Rombach. Experience factory. In J.J. Marciniak, editor, *Encyclopedia of Software Engineering*, volume 1, pages 469–476. John Wiley & Sons, New York, 1994.

[BM96] V.R. Basili and F.E. McGarry. The experience factory: how to build and run one, March 1996. Tutorial at the 18th International Conference on Software Engineering (ICSE).

[Fis98] G. Fischer. Seeding, evolutionary growth and reseeding: constructing, capturing and evolving knowledge in domain-oriented design environments. *Automated Software Engineering*, 5(4):447–464, October 1998.

[FLO+96] G. Fischer, S. Lindstaedt, J. Ostwald, K. Schneider, and J. Smith. Informaing system design through organizational learning. In *Proceedings on the 2nd International Conference on the Learning Society (ICLS)*, pages 52–59, Northwestern University, Evanston, 1996.

[HB99] F. Houdek and C. Bunse. Transferring experience: A practical approach and its application on software inspections. In *Proceedings on the SEKE Workshop on Learning Software Organizations*, pages 59–68, Kaiserslautern, Germany, June 1999.

[HSW98] F. Houdek, K. Schneider, and E. Wieser. Establishing experience factories at Daimler–Benz — An experience report. In *Proceedings of the 20th International Conference on Software Engineering (ICSE)*, pages 443–447. IEEE Computer Society Press, 1998.

[LS97] D. Landes and K. Schneider. Systematic analysis and use of experiences from software projects at Daimler-Benz. In A. Oberweis and H.M. Sneed, editors, *Software Management '97*, pages 63–73. Teubner Verlag, Stuttgart, 1997. (In German).

[LSH99] D. Landes, K. Schneider, and F. Houdek. Organizational learning and experience documentation in industrial software projects. *International Journal on Human-Computer Studies*, 51:643–661, 1999.

[WHS99] E. Wieser, F. Houdek, and K. Schneider. Systematic experience transfer: Three case studies from a cognitive point of view. In *Proceedings on the International Conference on Product Focused Software Process Improvement*, number 195 in VTT Symposium, pages 323–344, Oulu, Finland, June 1999.

Software Experience Center:
The Evolution of the Experience Factory Concept

Frank Houdek

DaimlerChrysler
Research and Technology
Dept. Software Engineering
frank.houdek@daimlerchrysler.com

Overview

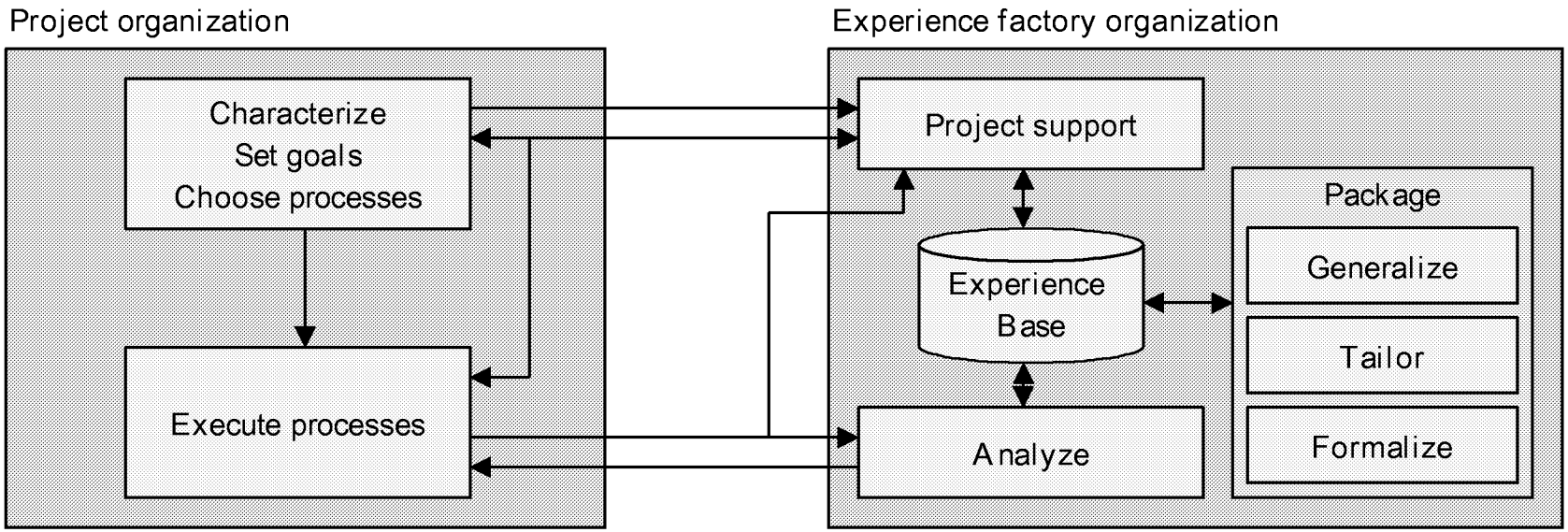
- Short company overview
- “Classic” experience factory - concepts and assumptions
- Dimensions of evolution
  - “quality of information”
  - reinfusion
  - seeding, evolution and reseeding
- Examples of evolved concepts
- Summary

Company Overview - DaimlerChrysler AG

- Main products:
  - Automobiles
  - Civil and military air- and spacecraft
  - Financial services
- "Experience factory history"
  - since 1997 experience factory initiatives in several business units (passenger car development, military aircraft, central IT services)
  - starting point: SEL experience factory concept
  - mission: establish an experience factory within 2-3 years
  - goal for initiatives: improve software development competency

Classic Experience Factory Concept

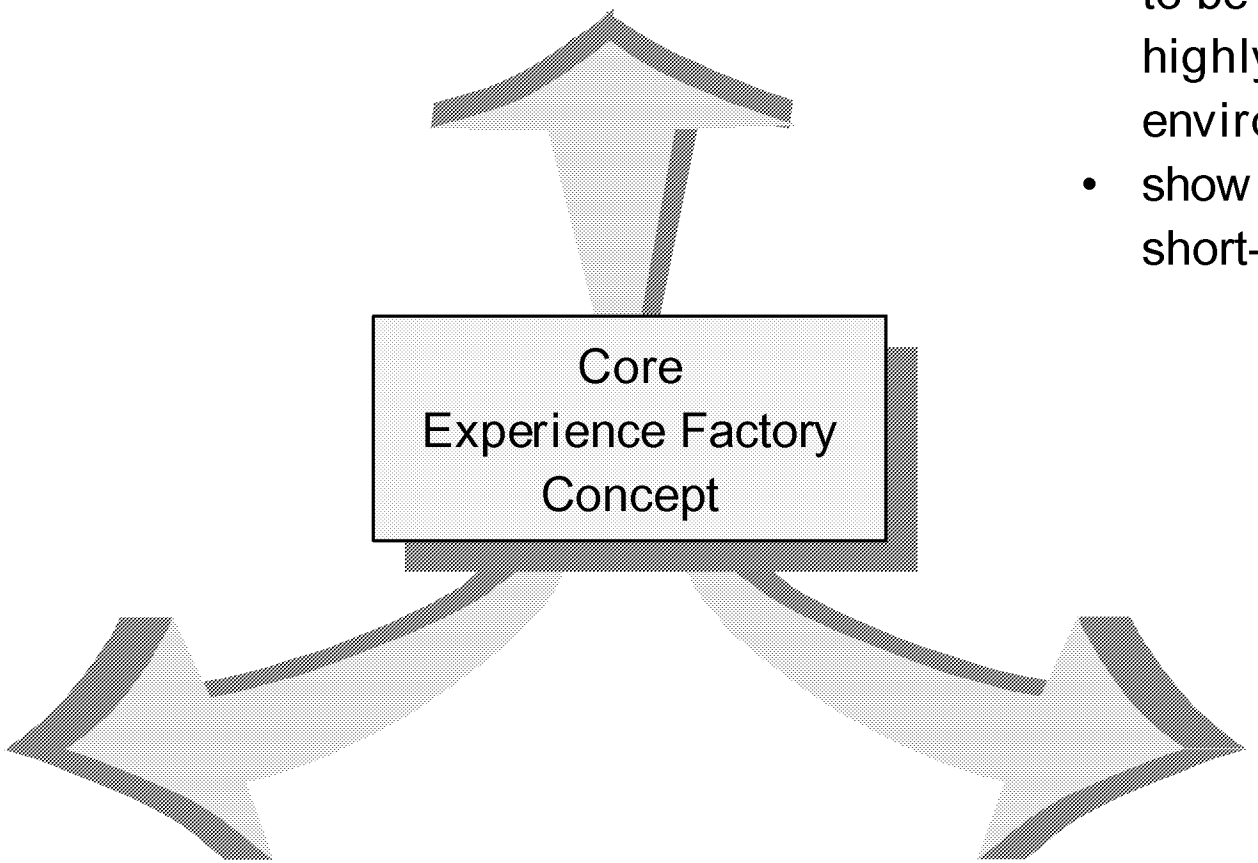
- QIP/GQM-based → long-term learning → "patience"
  → goal-oriented → "people know their needs"
- Reuse of models of all kinds
  (e.g. defect model, cost model, effort distribution, project plan, software architecture,
  } Explicit model buildung



Experience Factory: Assumptions and Reality

- More or less stable environment
  - some processes in place
  - some kind of "similarities" between projects (to reuse models)
  - common understanding of current situation, actual needs, and possible improvement mechanism
- "Homo economicus"
  - people are willing to accept new, better processes (i.e. pull)
  - people recognize long-term improvement efforts
- Constant management support
  - understanding about deficits and improvement mechanisms
  - encouragement of stuff

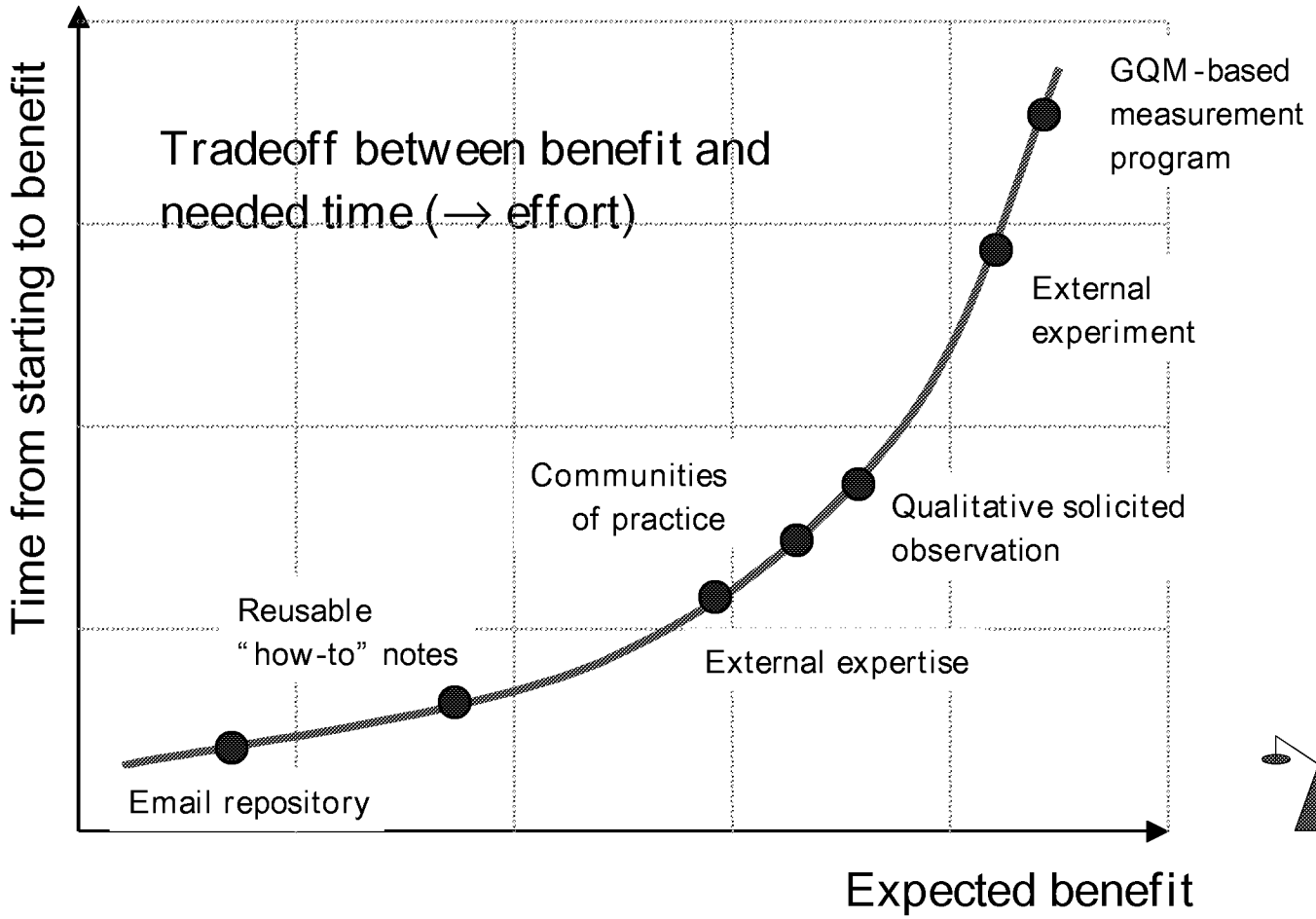
Dimensions of Evolution

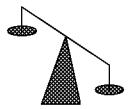


Main motivations:

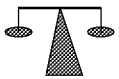
- to be more flexible in highly innovative environments
- show short-term benefits

„Quality of Information“



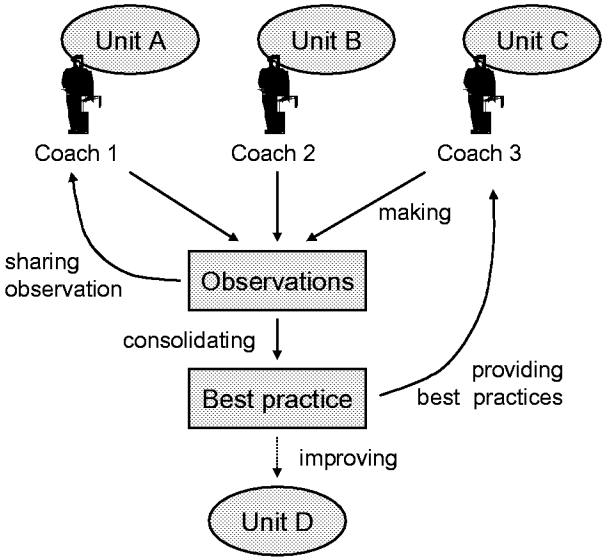


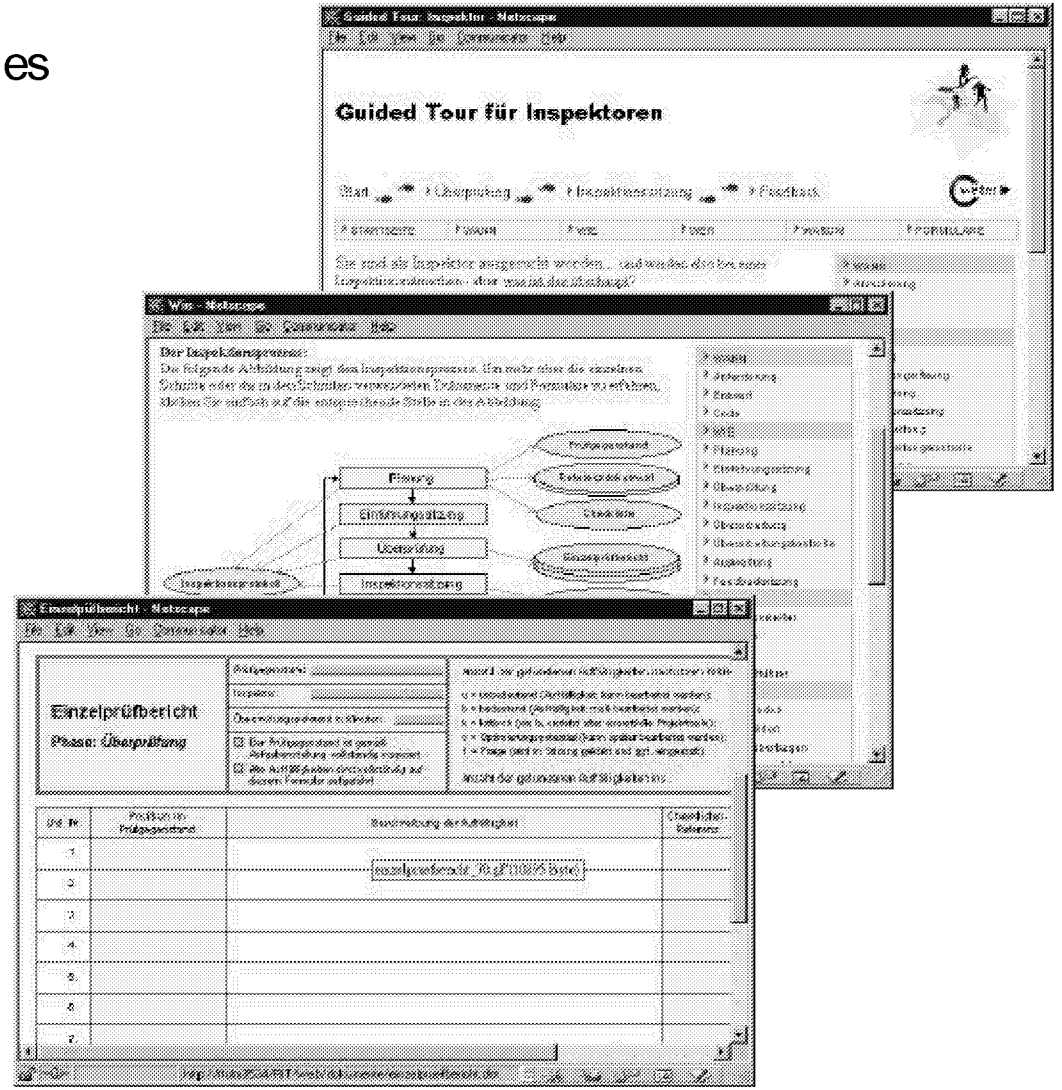
utility* = *benefit*/*effort



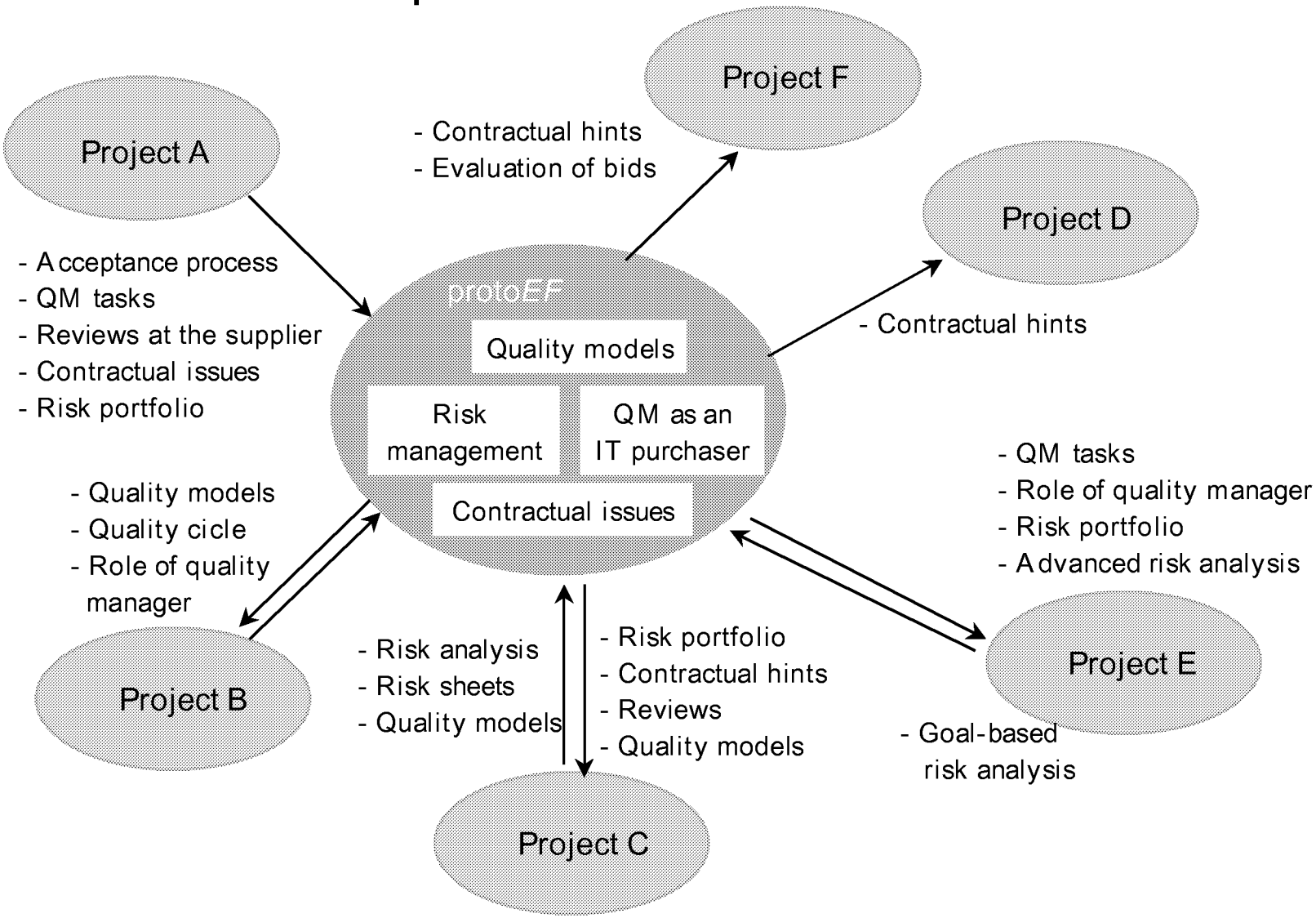
Example: Initial Seed, Evolution and Reseeding

- Experience base for review techniques
- Evolution of experience





Example: Reinfusion concepts



Central
IT services

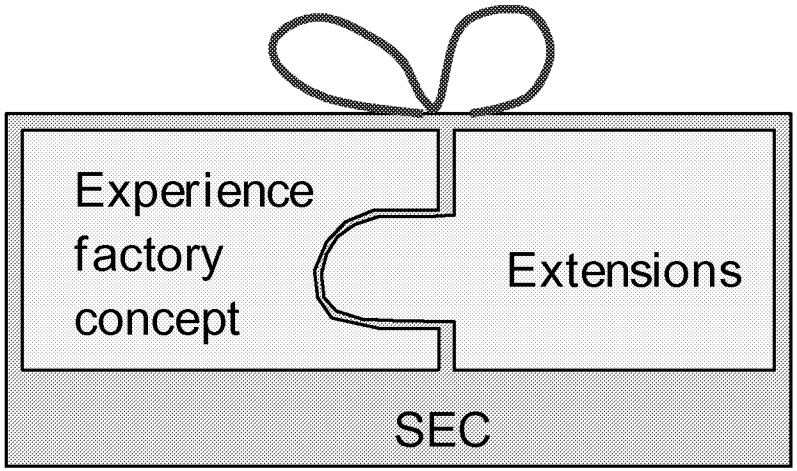
Summary

- SEL's EF concept is good starting point. However, tailoring is necessary
- Implementing the experience factory idea is a long term activity
- Short term improvements are expected

  → Experience factory concept has to provide suitable mechanisms

  → Providing a whole spectrum of mechanisms for
    - Experience reinfusion
    - Experience gaining
    - "Quality of Information"

→ Software Experience Center (SEC)



Session 2: Object Oriented Testing and Reading

Linda Rosenberg, SATC/Goddard

Melissa Major, Software Architects

Guilherme Travassos, University of Maryland

Risk-based Object Oriented Testing

Linda H. Rosenberg, Ph.D.	Ruth Stapko	Albert Gallo
NASA GSFC	SATC NASA, Unisys	SATC NASA, Unisys
Code 302	Code 300.1	Code 300.1
Greenbelt, MD 20771	Greenbelt, MD 20771	Greenbelt, MD 20771
301-286-0087	301-286-0101	301-286-8012
Linda.Rosenberg@gsfc.nasa.gov	Rstapko@pop300.gsfc.nasa.gov	Al.Gallo@gsfc.nasa.gov

Software testing is a well-defined phase of the software development life cycle. Functional ("black box") testing and structural ("white box") testing are two methods of test case design commonly used by software developers. A lesser known testing method is risk-based testing, which takes into account the probability of failure of a portion of code as determined by its complexity. For object oriented programs, a methodology is proposed for identification of risk-prone classes.

Risk-based testing is a highly effective testing technique that can be used to find and fix the most important problems as quickly as possible. Risk can be characterized by a combination of two factors: the severity of a potential failure event and the probability of its occurrence. Risk can be quantified by using the equation

$$\text{Risk} = \sum p(E_i) * c(E_i),$$

Where i =1,2,…,n. n is the number of unique failure events, E_i are the possible failure events, p is probability and c is cost.

Risk-based testing focuses on analyzing the software and deriving a test plan weighted on the areas most likely to experience a problem that would have the highest impact [McMahon]. This looks like a daunting task, but once it is broken down into its parts, a systematic approach can be employed to make it very manageable.

The severity factor $c(E_i)$ of the risk equation depends on the nature of the application and is determined by domain analysis. For some projects, this might be the critical path, mission critical, or safety critical sections. Severity assessment requires expert knowledge of the environment in which the software will be used as well as a thorough understanding of the costs of various failures. Musa addresses how to estimate the severity of software failures in the discussion of "Operational Profiles" in his book, *Software Reliability Engineering*. Both severity and probability of failure are needed before risk-based test planning can proceed. Severity assessment is not addressed here because it involves so much application-specific knowledge. Instead we confine the remainder of the discussion to the first part of the risk equation, ranking the likelihood of component failures, $p(E_i)$, and a way to capture the information directly from the source code, independent of domain knowledge.

The first task of risk-based testing is to determine how likely it is that each part of the software will fail. It has been proven that code that is more complex has a higher

incidence of errors or problems [Pfleeger]. For example, cyclomatic complexity has been demonstrated as one criterion for identifying and ranking the complexity of source code [McCabe]. Therefore, using metrics to predict module failures might simply mean identifying and sorting them by complexity. Then using the complexity rankings in conjunction with severity assessments from domain risk analysis would identify which modules should get the most attention. But module complexity is a univariate measure, and it could fail to detect some very risk-prone code. In particular, object oriented programming can result in deceptively low values for common complexity metrics. The nature of object oriented code calls for a multivariate approach to measure complexity [Rosenberg].

We are going to narrow the topic further and focus specifically on object oriented software. The Software Assurance Technology Center (SATC) at NASA Goddard Space Flight Center has identified and applied a set of six metrics for object oriented design measurement. These metrics have been used in the evaluation of many NASA projects and empirically supported guidelines have been developed for their interpretation. The metrics are defined as follows:

1. Number of Methods is a simple count of the different methods in a class.
2. The Weighted Methods per Class (WMC) is a weighted sum of the methods in a class [Chidamber]. If the weights are all equal, this metric is equivalent to the Number of Methods metric. The Cyclomatic Complexity [McCabe] is used to evaluate the minimum number of test cases needed for each method. Weighting the methods with their complexities yields a more informative class metric.
3. Coupling Between Objects (CBO) is a count of the number of other classes to which a class is coupled. It is measured by counting the number of distinct non-inheritance related class hierarchies on which a class depends [Chidamber]. Coupled classes must be bundled or modified if they are to be reused.
4. The Response for a Class (RFC) is the cardinality of the set of all methods that can be invoked in response to a message to an object of the class or by some method in the class [Chidamber].
5. Depth in Tree (DIT) - The depth of a class within the inheritance hierarchy is the number of jumps from the class to the root of the class hierarchy and is measured by the number of ancestor classes. When there is multiple inheritance, use the maximum DIT.
6. Number of Children (NOC) - The number of children is the number of immediate subclasses subordinate to a class in the hierarchy.

Having defined the metrics, we need interpretation guidelines to assist in identifying those areas of code deemed to be at high risk. For over three years, the SATC has been collecting and analyzing object oriented code written in both C++ and Java. Over 20,000 classes have been analyzed, from more than 15 programs. The results of the analyses have been discussed with project managers and programmers to identify threshold values

that do a good job of discriminating between "solid" code and "fragile" code.* Once the individual metric thresholds were determined, analysis revealed that a multivariate approach provided an excellent basis for planning risk-based testing.

When we first began to apply some of the traditional metrics to object oriented code, we saw that their values were generally much lower than we were accustomed to seeing for functionally written code. Judging by the old thresholds, the OO code appeared to be much less complex and much more modular than the non-OO legacy code. But because of the fundamentally different way an OO system is built, the low numbers were often very deceptive – ignoring the interactions between classes, and missing the complexities due to the use of inheritance. The following threshold values for the individual metrics were derived from studying the distributions of the metrics collected.

- Number of methods (NOM) - ≤ 20 preferred, ≤ 40 acceptable per class. The counting tool included explicit constructors and destructors in the method counts, so these thresholds are inflated. Taking that into account, the recommended number of actual implemented methods translates to under 10 per class.
- Weighted Methods per Class (WMC) - ≤ 25 preferred, ≤ 40 acceptable. The number of methods and the complexity of those methods are a predictor of how much time and effort is required to develop and maintain the class. While the NOM may be inflated by the beneficial use of constructors, WMC provides a better idea of the true total complexity of a class.
- Response for Class (RFC) - ≤ 50. We have seen very few classes with RFC over 50. If the RFC is high, this means the complexity is increased and the understandability is decreased. The larger the number of methods that can be invoked from a class through messages, the greater the complexity of the class, complicating testing and debugging. Making changes to a class with a high RFC will be very difficult due to the potential for a ripple effect.
- RFC/NOM ≤ 5 for C++, ≤. 10 for Java. This adjusted RFC metric does a good job of sifting out classes that need extensive testing, according to developer feedback. The Java language enforces the use of classes for everything, which automatically drives up the value of this metric.
- Coupling Between Objects (CBO) - ≤ 5. A high CBO indicates classes that may be difficult to understand, reuse or maintain. The larger the CBO, the higher the sensitivity to changes in other parts of the design and therefore maintenance is more difficult. Low coupling makes the class easier to understand, less prone to errors spawning, promotes encapsulation and improves modularity.
- Depth in Tree > 5 means that the metrics for a class probably understate its complexity. DIT of 0 indicates a "root"; the higher the percentage of DIT's of 2 and 3 indicate a higher degree of reuse. A majority of shallow trees (DIT's < 2) may represent poor exploitation of the advantages of OO design and inheritance. On the other hand, an abundance of deep inheritance (DIT's > 5) could be overkill, taking great advantage of inheritance but paying the price in complexity. When there is such

*It should be noted that the values of some of the OO metrics depend just as much on the design as they do on the actual coding. Much of the complexity of an OO system is fully determined before the programmers begin to write the code. Design complexity measurement is another topic that deserves researchers' attention.

liberal use of inheritance, the aforementioned class metrics will understate the complexity of the system.

- Number of Children (NOC) The greater the number of children, the greater the likelihood of improper abstraction of the parent and need for additional testing, but the greater the number of children, the greater the reuse since inheritance is a form of reuse. While there is no "good" or "bad" number for NOC, its value becomes important when a class is found to have high values for other metrics. The complexity of the class is passed on to all of its child classes and total system complexity is greater than it seemed at first glance.

A single metric should never be used alone to evaluate code risks, it takes at least two or three to give a clear indication of potential problems. Therefore, for each project, the SATC creates a table of high risk classes. High risk is identified as a class that has at least two metrics that exceed the recommended limits. Table 1 is an example of information that would be given to a project. The classes that exceed the expected limits are shaded.

Class	**# Methods**	**CBO**	**RFC**	**RFC/NOM**	**WMC**	**DIT**	**NOC**
Class 1	54	8	536	9.9	175	1	0
Class 2	7	6	168	24	71	4	0
Class3	33	4	240	7.2	105	2	0
Class7	54	8	361	6.7	117	2	2
Class8	62	6	378	6.1	163	2	0
Class 10	63	7	235	3.7	156	2	0
Class 11	81	10	285	3.5	161	2	0
Class 12	42	5	127	3.0	69	3	0
Class 14	20	17	324	16.2	139	4	4
Class 18	46	5	186	4.0	238	1	3

Table 1 : High Risk Java Classes

The purpose of the above information is to identify the classes at highest risk for error. While there is insufficient data to make precise ranking determinations, there is enough information to justify additional testing of classes which exceed the recommended specifications. It is up to the project to determine the criticality of these and the other classes to make the final determination on testing. Allocating testing resources based on these two factors, severity and likelihood of failures, amounts to risk-based testing.

Object oriented software metrics can be used in combination to identify classes that are most likely to pose problems for a project. The SATC has used the data collected from thousands of object oriented classes to determine a set of benchmarks that are effective in identifying potential problems. When problematic classes are also identified by domain experts as critical to the success of the project, testing can be allocated to mitigate risk. Risk-based testing will allow developers to find and fix the most important software problems earlier in the test phase.

References

Chidamber S.R. & Kemerer, C.F., "Towards a Metrics Suite for Object Oriented Design"" Proc. OOPSLA, 1991.

Li, W. & Henry, S., "Maintenance Metrics for the object Oriented Paradigm", 1st Int'l. Software Metrics Symposium, Baltimore MD, 1993.

McCabe, Thomas J., "A Complexity Measure*", IEEE Transactions on Software Engineering* SE-2, pp 308-320, 1976

McMahon, Keith, "Risk Based Testing", ST Labs, WA, 1998.

Pfleeger, S.L. and Palmer, J.D., "Software Estimation for Object Oriented Systems," Int'l. Function Point Users Group Fall conference, San Antonio TX, 1990

Rosenberg, Linda, and Gallo, Albert, "Implementing Metrics for Object Oriented testing", Practical Software Measurement Symposium, 1999.

"Risk-Based Object Oriented Testing"

Software Assurance Technology Center

Goddard Space Flight Center - NASA

http://satc.gsfc.nasa.gov

Dr. Linda H. Rosenberg
301-286-0087
Linda.Rosenberg@gsfc.nasa.gov

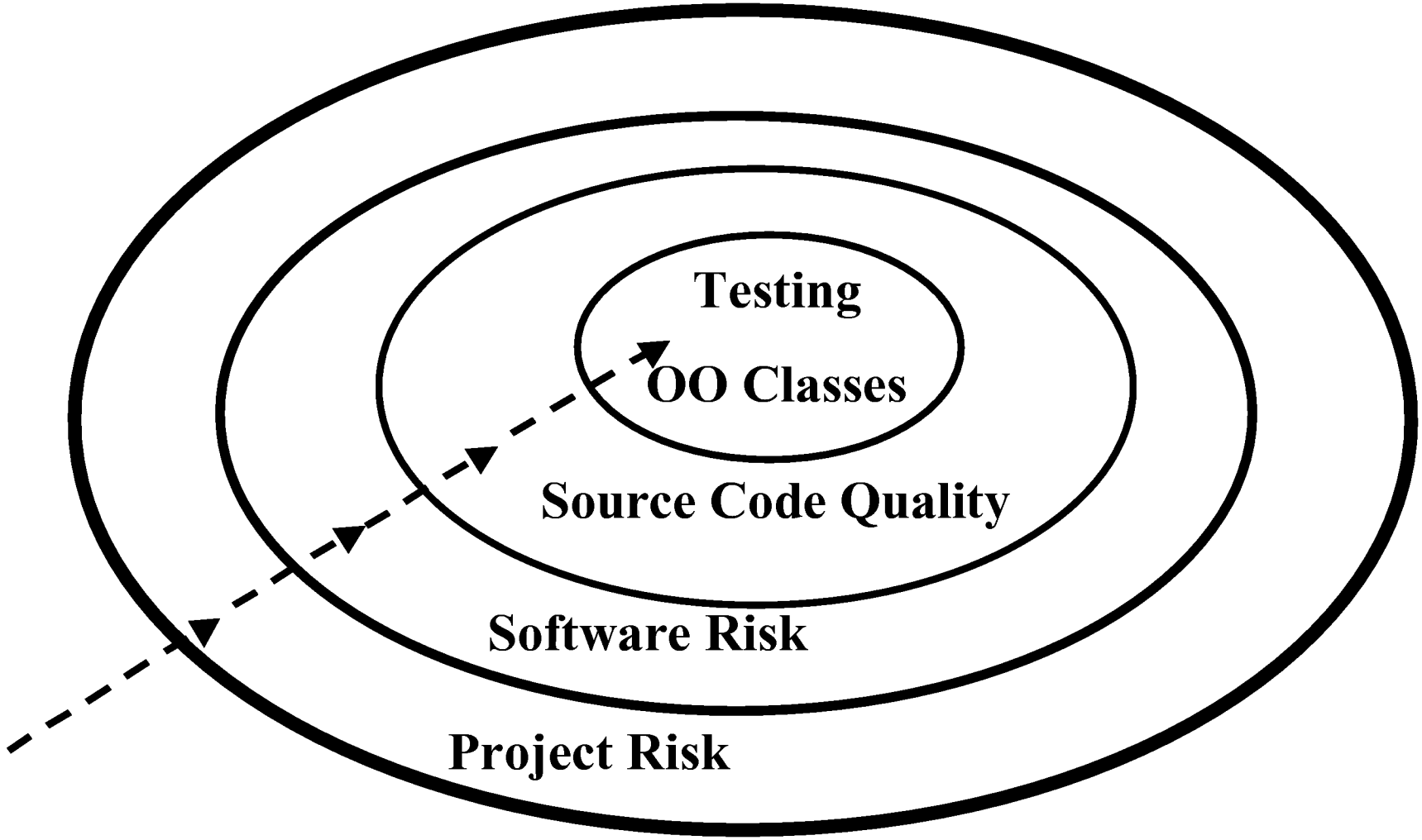
Ruth Stapko
301-286-0101
rstapko@pop300.gsfc.nasa.gov

Al Gallo
301-286-8012
Al.Gallo@gsfc.nasa.gov

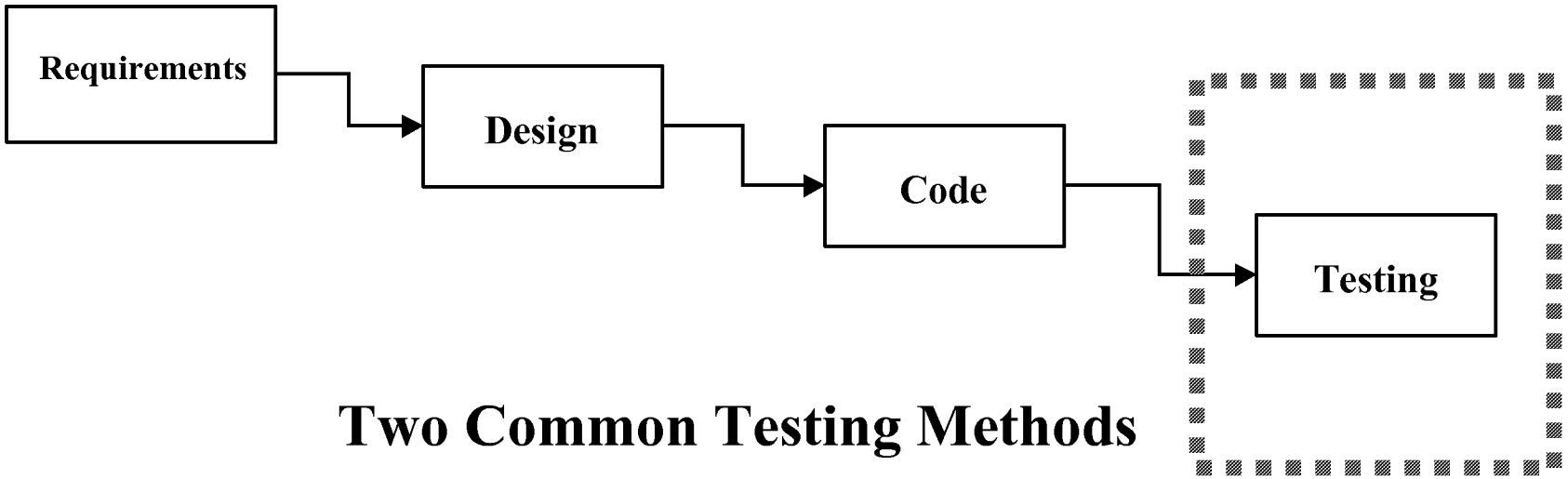
Michael S. Parizer
301-286-7297
mparizer @pop300.gsfc.nasa.gov

NASA SATC
Software Assurance
Technology Center

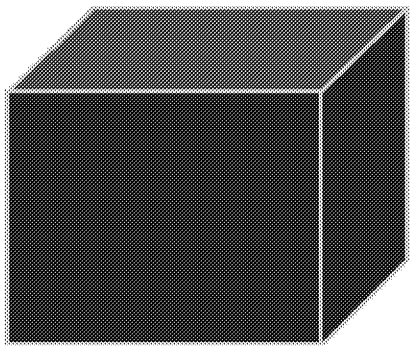
Overview



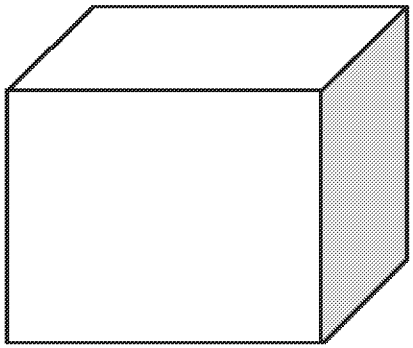
Testing Approaches



Two Common Testing Methods



Black Box

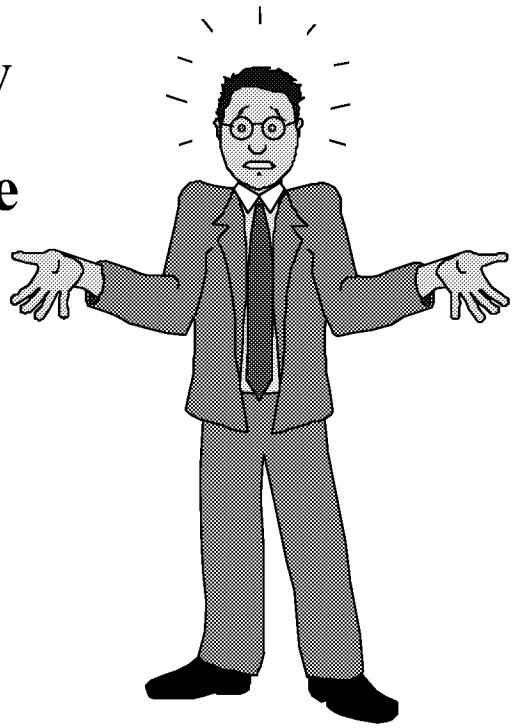


White Box

Risk - Based Testing

What Does Risk-Based Testing Do For Me?

It focuses on analyzing software and deriving a test plan weighted on the areas most likely to experience a problem that would have the highest impact.



Risk-Based Testing

- **Project Risk is characterized by two features:**
  - **1) The probability of a potential failure event**
  - **2) The severity of its occurrence**

- **Risk is quantified by using the following equation:**

$$\text{Risk} = \sum p(E_i) * c(E_i)$$

E_i is the i-th possible failure event

p is the probability the event will occur

c is cost of an event if it does occur

Risk-Based Testing

Cost/Severity factor $c(E_i)$:

- Depends on the nature of the application
- Determined by domain analysis
- Requires expert system knowledge

Risk-Based Testing

Software Risk

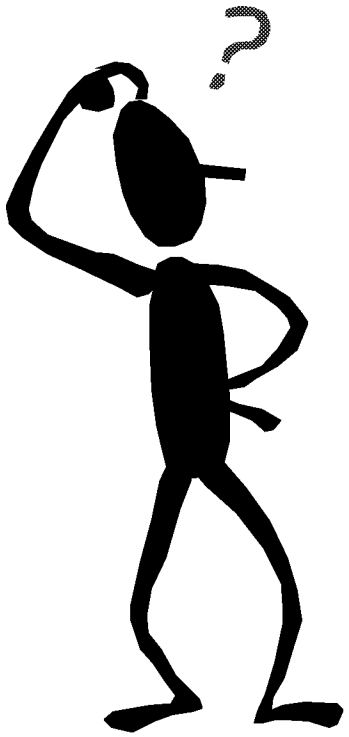
1. **Let c (E_i) represent the cost of a failure in a particular component *of the software***

2. **Use source code analysis to rank likelihood of failure, p (E_i):**

   - **Code that is more complex has a higher incidence of errors**
   - **Example: Cyclomatic Complexity is a measure for ranking the complexity of source code.**

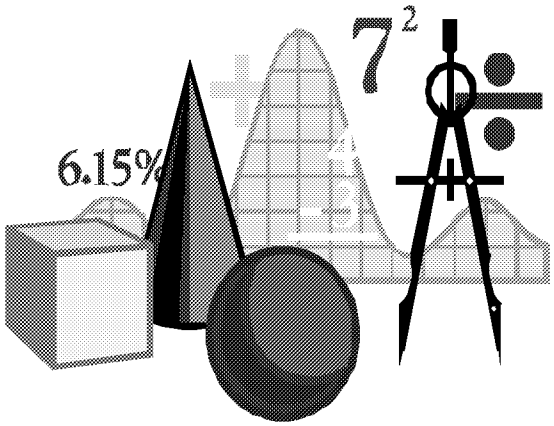
Object Complexity

- **Cyclomatic complexity measures standalone risk**
  - **Can result in deceptively low values**
- **Object classes are not isolated!**
  - **Inheritance**
  - **Message passing**



Identification of High Risk Classes

Several metrics are available for comparing OO classes:



1. **Number of Methods (NOM)**

   A simple count of methods in a class definition

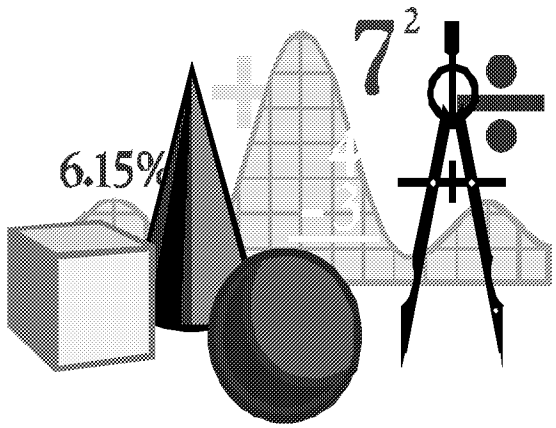
2. **Weighted Methods per Class (WMC)**

   The sum of the cyclomatic complexities of the methods in a class

3. **Coupling Between Objects (CBO)**

   A count of other classes "bundled" with a class (other than through inheritance)

Identification of High Risk Classes



4. Response for a Class (RFC)

Count of all methods that an object of the class could invoke. A "worst case" metric.

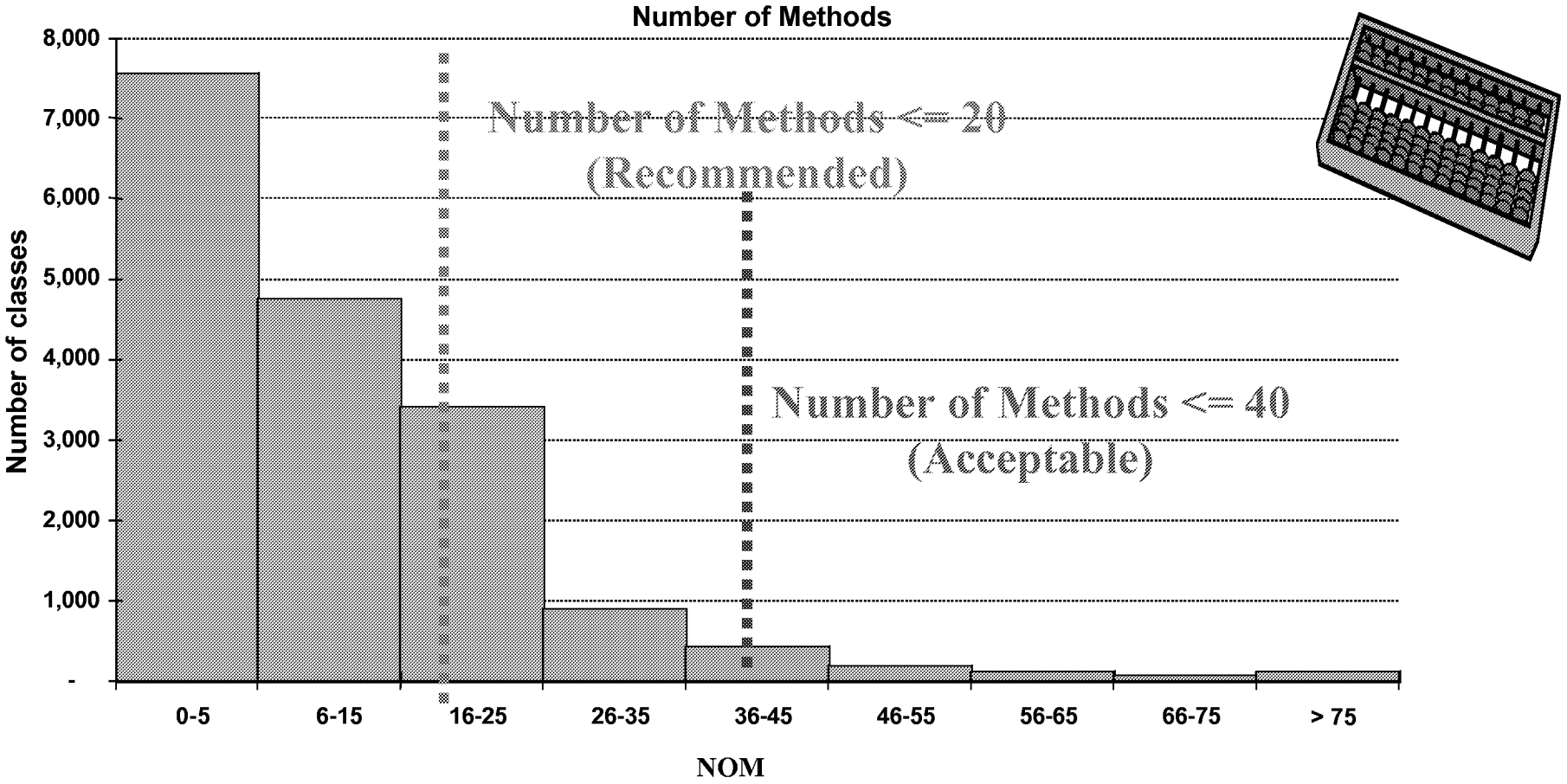
5. Depth in Tree (DIT)

How many levels is the class down from the top? Measures "hidden" complexity.

6. Number of Children (NOC)

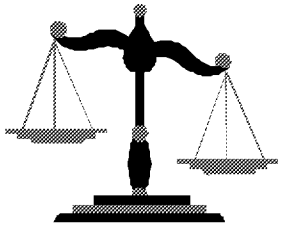
Child classes take advantage of reuse, but all will suffer if parent classes are too complex.

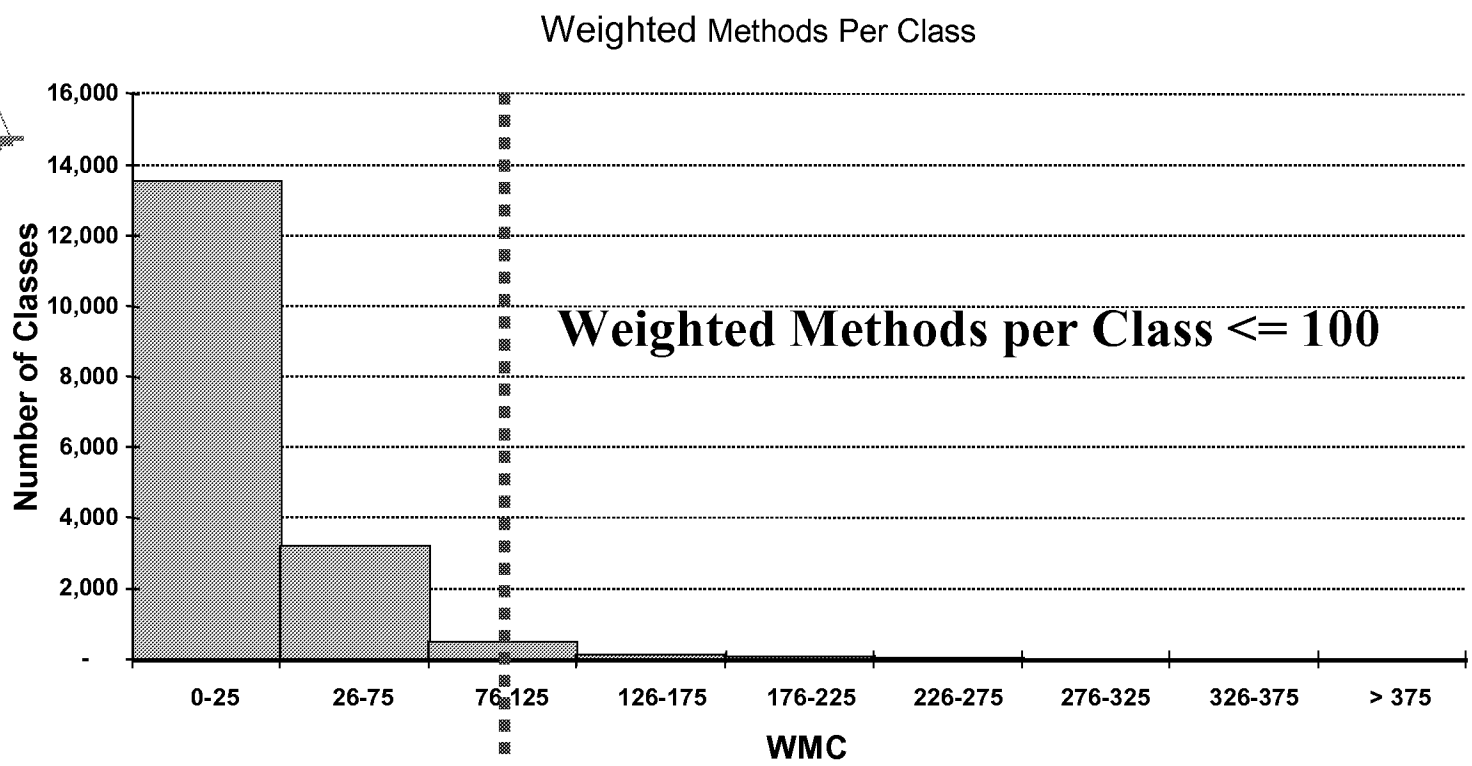
Number of Methods per Class



NOM measures both size and complexity of a class. It may be necessary to trade off some efficiency to preserve maintainability.

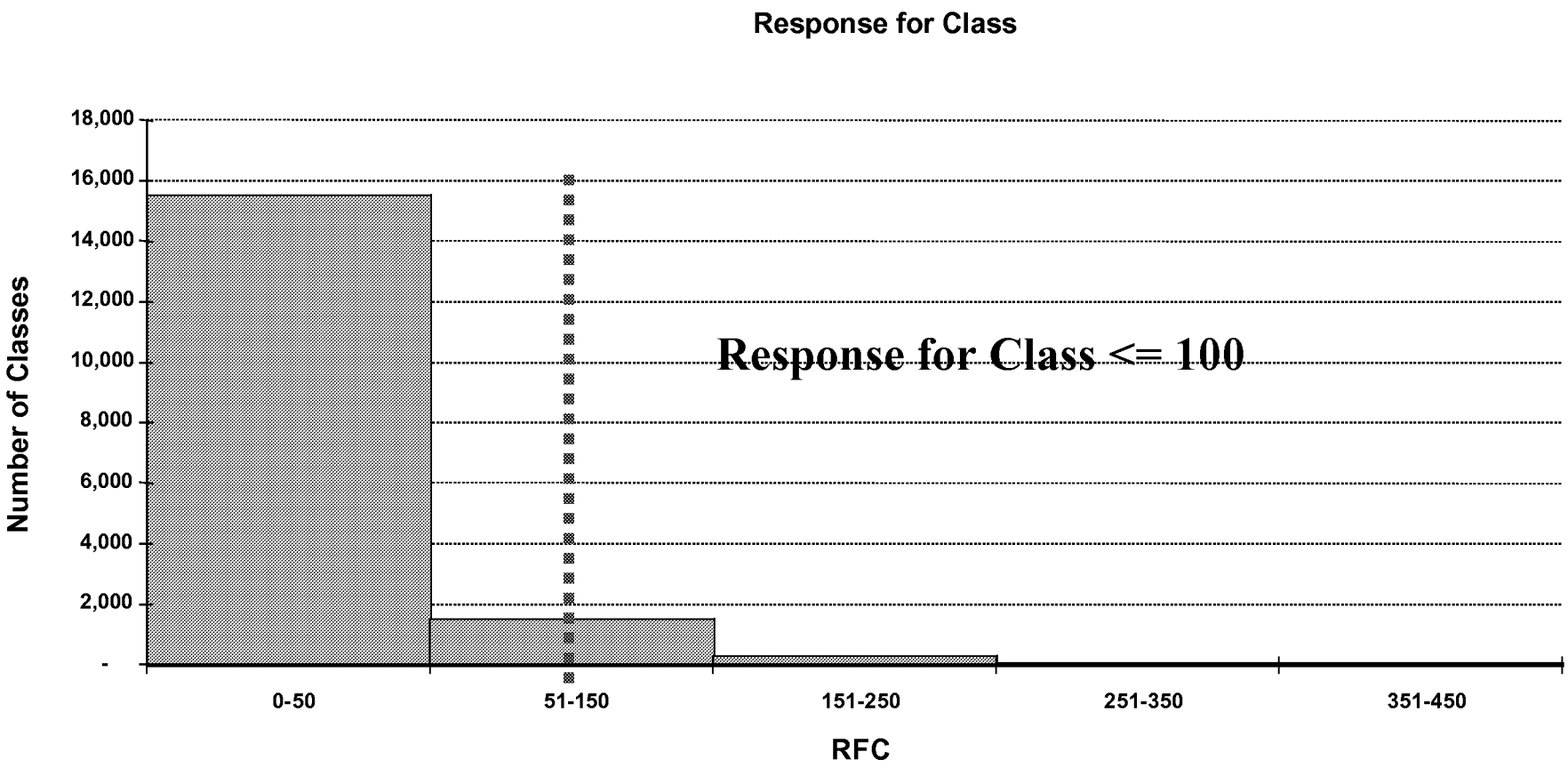
Weighted Methods per Class





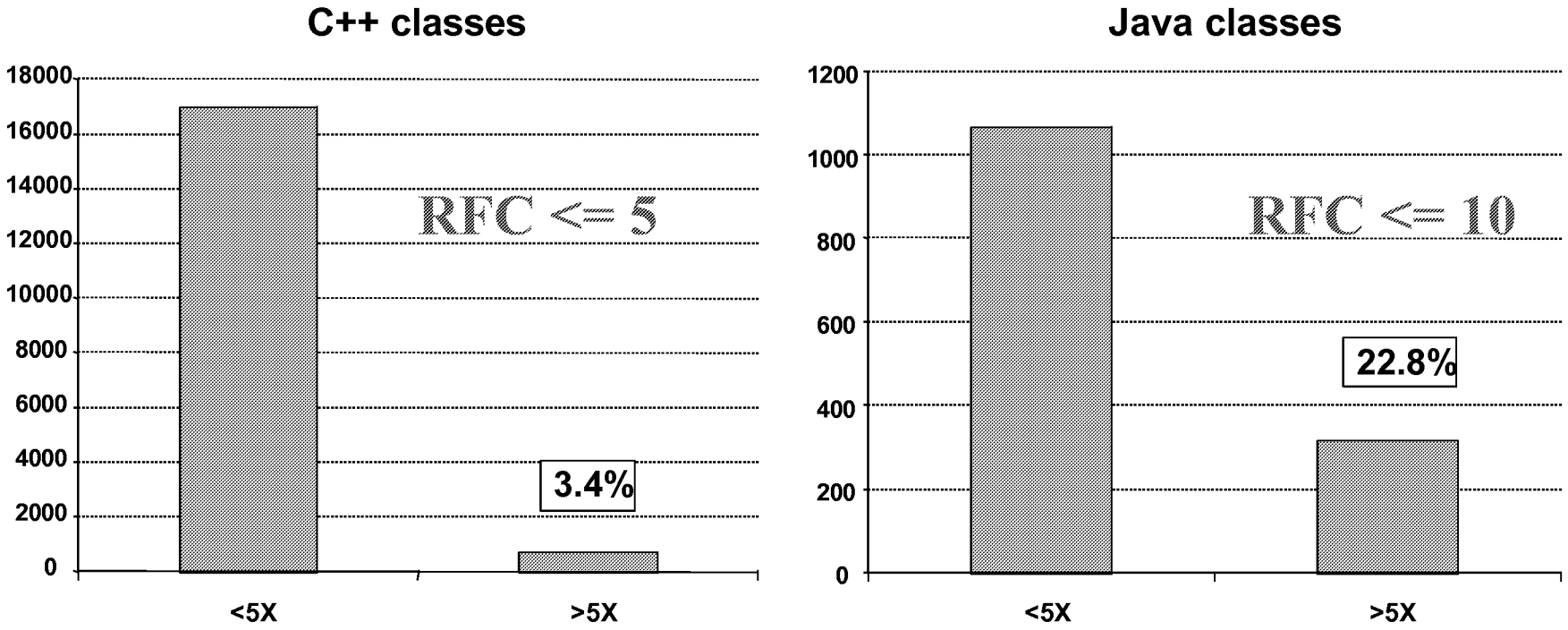
WMC is defined here as the sum of the cyclomatic complexities of the methods implemented in one class.

Response for Class



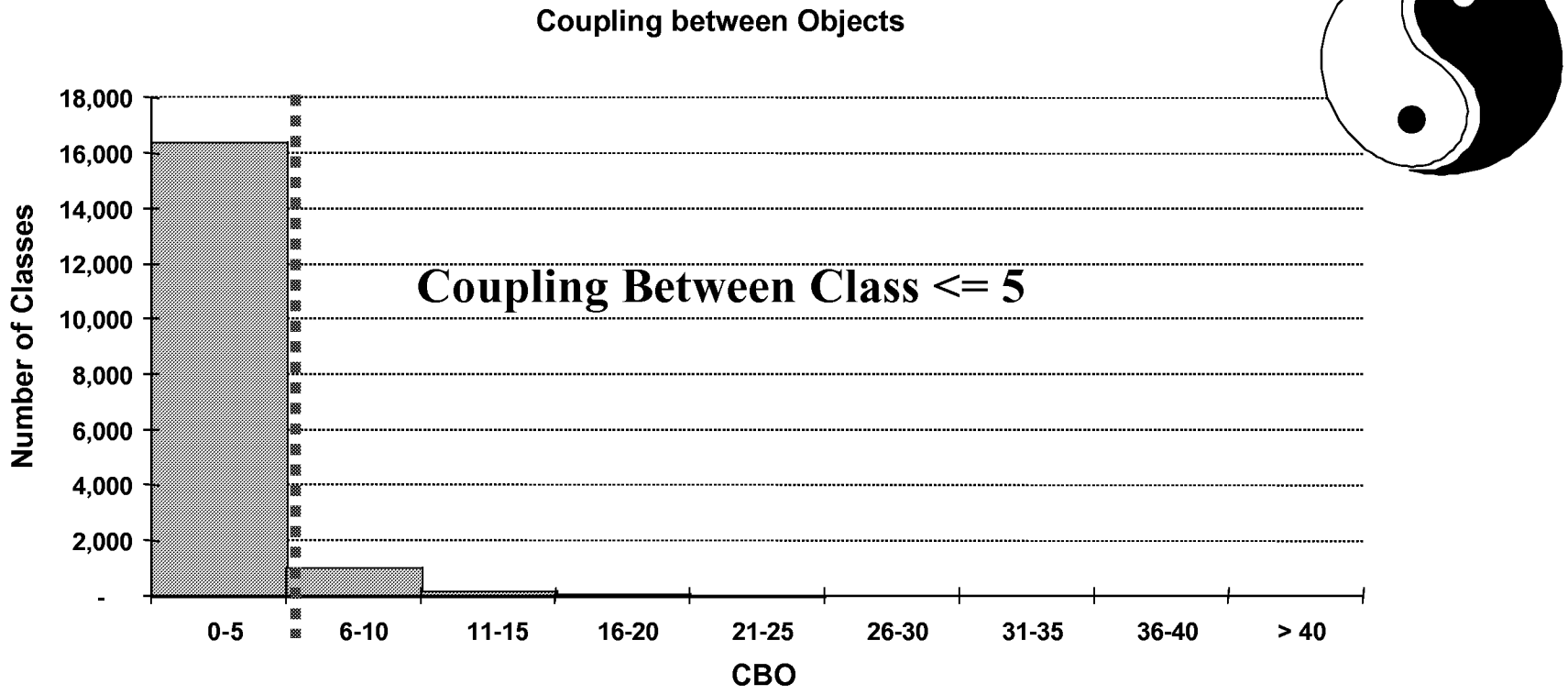
If RFC is high, the codes complexity is increased and its understandability is decreased.

Response for Class ÷ by NOM



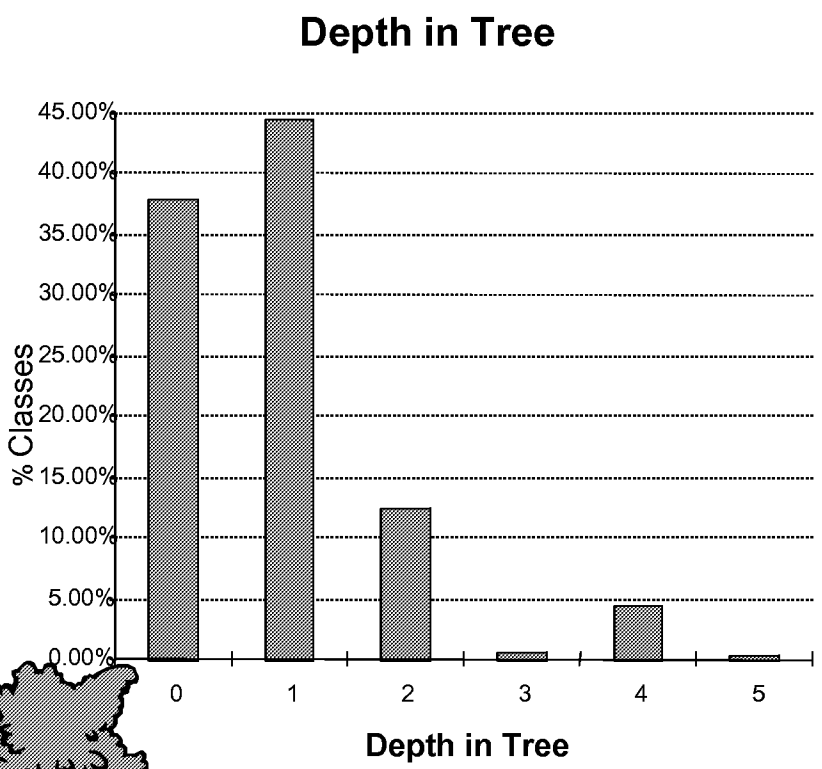
Experience tells us that this derived metric does a good job of finding classes that require extra testing. Java code tends to have higher RFC values.

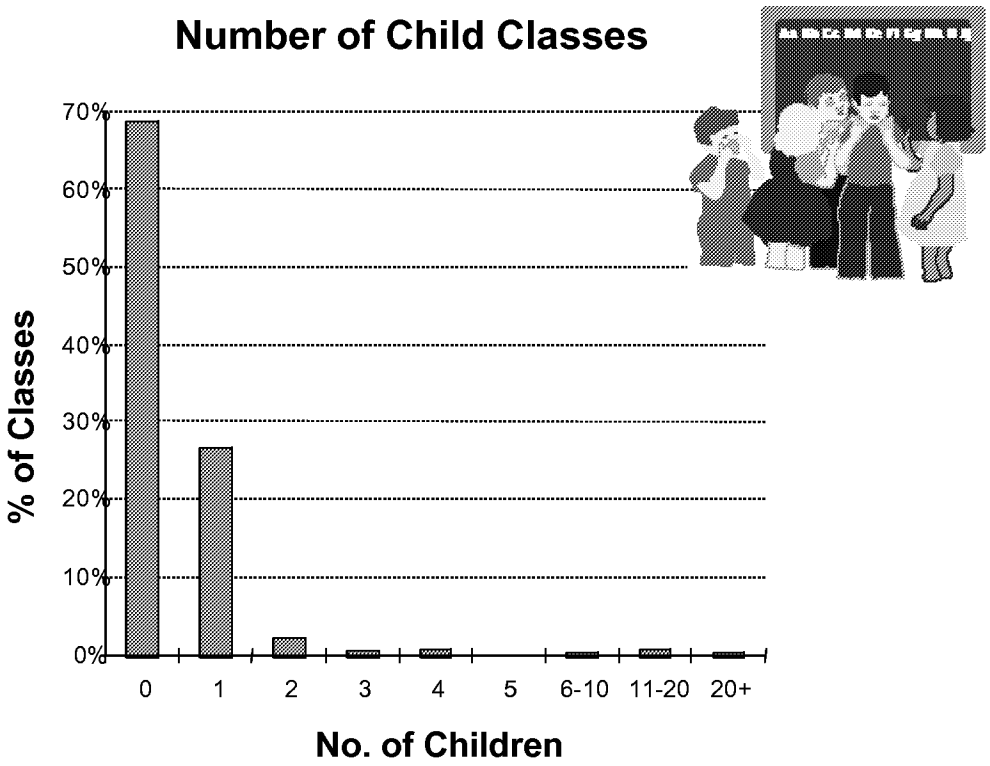
Coupling Between Objects

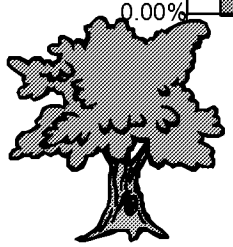


Coupling is a measure of inter-class complexity, a design issue. The larger the CBO, the more sensitivity to changes, maintenance is more difficult.

Depth in Tree & Number of Children





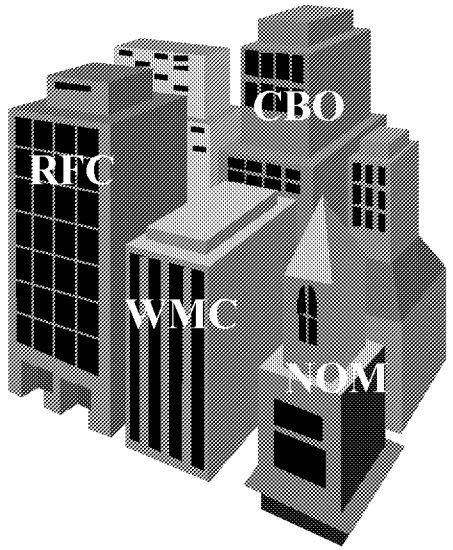


Deeply nested inheritance may hide complexity. The greater the NOC, the more likely the child classes will have improper abstraction.

So Many Metrics . . .

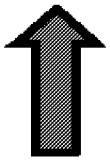
Class	# Methods	CBO	RFC	RFC/NOM	WMC	DIT	NOC
Class 1	54	8	536	9.9	175	1	0
Class 2	7	6	168	24	71	4	0
Class 3	33	4	240	7.2	105	2	0
Class 7	54	8	361	6.7	117	2	2
Class 8	62	6	378	6.1	163	2	0
Class 10	63	7	235	3.7	156	2	0
Class 11	81	10	285	3.5	161	2	0
Class 12	42	5	127	3	69	3	0
Class 14	20	17	324	16.2	139	4	4
Class 18	46	5	186	4	238	1	3

A single metric should never be used alone, look for classes that have more than one high value.



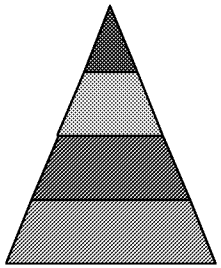
Risk - Based Testing - Summary

Define High Risk



Identify components of High Risk

Rank High Risk components



Plan Extra Testing in High Risk areas

NASA SATC
Software Assurance Technology Center

Future Work

- **Refine threshold values by programming language, program size, application domain.**

- **Use multivariate techniques to find one or two OO complexity metrics that quantify class risk**

- **Post-hoc validation of risk-based testing**

References

Chidamber S.R. & Kemerer, C.F., "Towards a Metrics Suite for Object Oriented Design" Proc. OOPSLA, 1991

Li, W. & Henry, S., "Maintenance Metrics Suite for Object Oriented Paradigm", 1st International Software Metrics Symposium, Baltimore MD, 1993

McCabe, Thomas J., "A Complexity Measure", *IEEE Transactions on Software Engineering* SE-2, pp 308-320, 1976.

McMahon, Keith, "Risk Based Testing", ST Labs, WA, 1998.

Pfleeger, S.L. and Palmer, J.D., "Software Estimation for Object Oriented Systems", International Function Point Users Group Fall Conference, San Antonio TX, 1990.

Rosenberg, Linda, and Gallo, Albert, "Implementing Metrics for Object Oriented Testing", Practical Software Measurement Symposium, 1999.

Using Guided Inspection to Validate UML Models

Melissa L. Major and John D. McGregor
Software Architects

Guided inspection is an inspection/review technique that is "guided" by test cases. Inspections are used to provide a detailed examination of a design or program by a human, as opposed to a machine's execution of a prototype or completed application. However, even Fagan-style inspection processes focus more on the form of the inspection process rather than the substance of the material being inspected. Standard inspection techniques also focus on examining what is in the inspection material rather than determining whether there is something that is missing from the model or code.

These standard inspections are often a top down reading of the code or a scan of a diagram. The top down approach makes the measurement of coverage straightforward but it is more difficult for the inspector to ensure that appropriate connections have been made between objects. The use of test cases means that the inspection process can address more than just the syntax of the diagram or code being reviewed. The test cases come from test plans that are already a required part of the software development process.

Techniques such as checklists have been used to summarize the results of an inspection and to ensure that the inspector does a thorough job. Guided inspection supplements the checklist with the testing concept of "coverage". Coverage measures determine how much of the product being inspected has been examined. Test cases are selected from the test plan so that, for example, every use case is represented by at least one test case.

Studies have reported widely varying savings ratios for finding faults early in the development process as opposed to during the compilation or system test phases. For example, IBM reported that repairing a fault found at system test time may cost as much as 100 times the cost of repairing the same fault found during design. With this amount of margin even a technique that is relatively expensive can still result in time and cost savings.

The Testing Perspective Applied to Inspections

Applying a testing approach to inspections provides several benefits. In this section we examine these benefits and provide guidelines for the inspection process that ensure these benefits are realized.

Objectivity

For testing to be effective it must be conducted objectively. A person testing their own code is seldom sufficiently objective to achieve optimum results. If the person has made a wrong interpretation of the inputs to their process, that mistake will simply be carried forward into the test cases. A second person, or even an automated tool, will provide a different, although not always correct perspective.

Guideline #1 – ***Select inspectors from outside the immediate development team.***

Traceability

For testing materials to be maintainable it must be easy to map changes in the model to needed changes in the test scenarios. In an iterative development environment changes occur frequently to all parts of the project. Changes in requirements are reflected in changes to the use cases. Changes in class specifications should signal the need for regression testing of the effected parts of the model.

Guided inspection uses scenarios derived from the use case description as the primary test case description. A project should maintain a matrix that associates a package with all the use cases for which that package is needed. Then each time changes are made to a package, the affected use cases and scenarios are easily identified.

Guideline # 2 – ***Maintain a mapping between use cases and the classes/packages that realize those use cases.***

Testability

For testing to be possible, the model must be testable. This implies that the model is sufficiently specific to support the evaluation of test execution results. Domain models are general by design and there is a fine line between vague generality and sufficient detail to support testing.

Guideline # 3 – ***Assign a team member to write test cases as the modeling proceeds. Have the "testing" domain expert review these. Feedback, into the modeling process, any information indicating places where the model is too vague.***

This is common advice that we give to process definers at all levels. There should be a validation activity for each development phase. Preparation for that activity should proceed in parallel with the development activity. This allows the act of preparation to actually help improve the product before the formal validation. Writing test cases is an excellent technique for providing continuous feedback during development.

Coverage

For testing to provide us with confidence, we need to know how thoroughly the product under test has been examined. The general term for this type of metric is ***coverage***. When we speak of "functional" testing, we mean that the coverage will be expressed in terms of the functional specification of the product under test. The metric is chosen to give some notion of completeness at the appropriate level.

For guided inspection there are two different possible bases for coverage: the class/state/activity diagrams and use case diagrams. The use case diagram is a good source of scenarios; however, we are more concerned that the domain model contains a complete set of concepts for the domain. These are represented in the class diagram and further clarified in the state and activity diagrams for each class.

Guideline # 4 – ***Use copies of the model's diagrams and mark off each element in a diagram as it is used in a test scenario.***

Developing a test scenario for each actor in the use case diagram is a minimal level of coverage. One scenario per primary use case is a stronger coverage criterion. Covering every primary use and then adding coverage for all "alternate courses of action" for use cases that are rated frequent and critical is an even stronger criterion. Once the set of scenarios are run through the model, the resulting coverage of the class diagram and state diagrams provides a check of the thoroughness with which the model has been inspected.

Criteria for a Good Model

The Guided Inspection evaluation criteria used by models are described more completely in [1]. They are:

- correctness
- completeness
- consistency

Correctness is a measure of how accurately the model represents the information. Correctness of the model is really the aggregate of judgements from the individual test cases. Each test case includes a description of the results expected from executing the test case. This expected result is based on a source that is assumed to be (nearly) infallible, a "test oracle". The oracle usually is a human expert whose personal knowledge is judged to be sufficiently reliable to be used as a standard. The tester judges the accuracy of the model's representation of concepts relative to the results expected by the oracle.

A model is correct with respect to a test case if the result of the execution is the result that was expected. A model is correct if each of the test cases produces the expected results. The problem here is whether the

"expected" result really is the appropriate one. In the real world, we must assume that the oracle can be incorrect on occasion.

Completeness is a measure of whether a necessary, or at least useful, element is missing from the model. It is judged by determining if the entities in the model describe the information being modeled in sufficient detail for the goals of the current portion of the system being developed. This judgement is based on the model's ability to represent the required situations and on the knowledge of experts. In an iterative incremental process, completeness is considered relative to how mature the current increment is expected to be. This criterion becomes more rigorous as the increment matures over successive iterations.

One factor directly affecting the effectiveness of this criterion is the quality of the test coverage. The model is judged complete if the results of executing the test cases can be adequately represented using only the contents of the model. For example, a sequence diagram might be constructed to represent a scenario. All of the objects needed for the sequence diagram (SD) must come from classes in the class diagram or it will be judged incomplete. However, if only a few test cases are run, missing classes may escape detection. In most cases, this type of testing is sufficiently high level that coverage of 100% is achievable and desirable.

Consistency is a measure of whether there are contradictions among the various diagrams within the model and between models produced during various phases. This may be partially judged by considering whether the relationships among the entities in the model allow a concept to be represented in more than one way. For example, each name should be unique. In an incremental approach the consistency is judged locally until this increment is integrated with the larger system. The integration process must ensure that the new piece does not introduce inconsistencies into the integrated model.

Consistency checking can determine whether there are any contradictions or conflicts present either internal to a single diagram or between two diagrams. For example, one diagram, perhaps a sequence diagram, might require a relationship between two classes while another diagram, such as the class diagram, shows none. Inconsistencies will often initially appear as incorrect results in the context of one of the two diagrams and correct results in the other. Inconsistencies are identified by careful evaluation of the results of a simulated execution.

A Basic Process

Roles

There are several roles in this process. Several roles may be assigned to a single person; however, to ensure objectivity there should be a clear distinction between the producers of the model under test (MUT) and the testers/inspectors.

> **Test oracle** - These personnel are the source of truth (or at least expected test results). They define the expected system response for a specific input scenario. These will usually be either domain experts or system engineers.
>
> **Test case writer** - These personnel perform the analysis necessary to select test cases. They also record the expected result for each test case as defined by the Oracle. These people may be developers who did not create the model or system test personnel.
>
> **Symbolic executioner** - These personnel provide the actual system response as defined to this point in the software development process. These will typically be members of the team developing the MUT since they understand the operation of the individual elements of the model.
>
> **Moderator** - The Moderator controls the session and advances the discussion through the scenario.

Recorder - This person makes modifications to the reference models as the team agrees upon changes. The Recorder also makes certain that these changes are taken into consideration in the latter parts of the scenario. The Recorder also maintains a list of issues to record questions that are not resolved during the testing session.

Drawer - This person constructs the SD as the scenario is executed. He/ she concentrates on capturing all of the appropriate details such as returns from messages and state changes. They may also annotate the SD with information between the message arrow and the return arrow.

Steps

The model testing process is tightly coupled with the model development process [2]. We have found it useful to iterate within the modeling process by periodically switching from the modeling activity to the testing activity. This provides quick feedback and often provides new information to be modeled.

The basic steps are the same as for any testing process:

- **Analyze** - Much of the testing analysis has been done if the use case descriptions contain sufficient information to allow them to be prioritized. We use a weighted frequency profile to prioritize use cases for testing. The weight is based on how critical the use is to the success of the system.
- **Construct** - Write scenarios from the use cases. Each scenario must be made more specific by providing exact values for attributes. These values are selected by first establishing equivalence classes of values. Equivalence classes of values are all values that will provide the same behavior in a given context. For example, {0,1,2,3} all produce the same response from the statement `x > -1 and x<4`. Many of the test parameters will be objects. Equivalence class translates to "objects that are in the same state no matter how they got there." Different use cases will have different numbers of test cases. We select from some states more frequently than others due to their participation in high priority use cases.
- **Execute and Evaluate** – The inspection process combines the application of a checklist with the execution of test cases. The test session involves role-playing in which the modelers and developers step through the model. The test session is a group meeting since no individual developer will understand all of the classes in the model and few models contain all the relevant information. The moderator selects one of the scenarios and triggers the use. The developer/ owner of the class that begins the scenario by describing the action taken by his/her object and describes its interaction with other objects. For each interaction, the owner of the class receiving the message describes their interaction with other objects and execution proceeds along each of these links.

Summary of Our Experience

The Guided Inspection technique has been used in a variety of forms on a number of projects that differ in size, complexity and domain. The technique has been used in the usual analysis and design contexts where a development organization applied the technique to each model produced by an increment team. It has also been used in limited engagements where a domain model or an architectural model was the only artifact being evaluated. Our experience and that of knowledgeable clients is that this technique has greater defect finding power than other widely used inspection techniques. The technique does require more effort (the construction of test cases) than other inspection techniques but it is effort that would be expended anyway.

The use of test cases brings a logical continuity to the inspection. Each step in the test case is a logical consequence (rather than a syntactic necessity) of the previous steps. This guides the inspectors through the material to be inspected in a path that allows them to judge the semantic validity of the model in

addition to evaluating its syntactic correctness. The result is that the defects that are found have the potential of greater impact on the system than the syntactic bugs found in a sequential search.

Conclusion

We have presented an overview of guided inspection. This quality technique provides a means for examining models and code in a semantically meaningful way rather than examining disjoint pieces of syntax. Detecting defects in the early analysis and design models makes a major contribution to the quality of the application and to an on-time, on-budget delivery. Our presentation at the workshop will elaborate on the steps in the basic process and illustrate the models being inspected.

References

1. John D. McGregor. *The Fifty Foot Look at Analysis and Design Models*, **Journal of Object-Oriented Programming**, July/August 1998.
2. John D. McGregor. *Testing Models, The Requirements Model*, **Journal of Object-Oriented Programming**, June 1998.

A Process Definition for Guided Inspection

John D. McGregor
Melissa L. Major
Software Architects

Goal: To identify defects in artifacts created during the analysis and design phases of software construction.

Steps in the Process

1. Define the scope of the Guided Inspection
2. Identify the basis model(s) from which the material being inspected was created
3. Assemble the GI team
4. Define a sampling plan and coverage criteria
5. Create test cases from the bases
6. Apply the checklist and tests to the material
7. Gather and analyze test results
8. Report and feedback

Detailed Step Descriptions

Define the scope of the Guided Inspection

Inputs:

The project's position in the life cycle.
The materials produced by the project (UML models, plans, use cases).

Outputs:

A specific set of diagrams and documents that will be the basis for the evaluation.

Method:

Define the scope of the GI to be the set of deliverables from a phase of the development process. Use the development process information to identify the deliverables that will be produced by the phase of interest.

Example:

The project has just completed the domain analysis phase. The development process defines the deliverable from this phase as a UML model containing domain-level use cases, static information such as class diagrams and dynamic information such as sequence and state diagrams. The GI will evaluate this model.

Identify the basis model(s) from which the material being inspected was created

Inputs:

The scope of the GI.
The project's position in the life cycle.

Outputs:

The material from which the test cases will be constructed (The Model Under Test - MUT)

Method:

Review the development process description to determine the inputs to the current phase. The basis model(s) should be listed as inputs to the current phase.

Example:

The inputs to the domain analysis phase is the "knowledge of experts familiar with the domain". These mental models are the basis models for this GI.

Assemble the GI team

Inputs:

The scope of the GI.
Available personnel.

Outputs:

A set of participants and their roles.

Method:

Assign persons to fill one of three categories of roles: Administrative, Participant in creating the model to be tested, Objective observer of the model to be tested. Choose the objective observers from the customers of the model to be tested and the participants in the creation of the basis model.

Example:

Since the model to be tested is a domain analysis model and the basis model is the mental models of the domain experts, the objective observers can be selected from other domain experts and/or from application analysts. The creation participants are members of the domain modeling team. The administrative personnel can perhaps come from other interested parties or an office that provides support for the conduct of GIs.

Define a sampling plan and coverage criteria

Inputs:
The project's quality plan.

Outputs:
A plan for how test cases will be selected.
A description of what parts of the MUT will be covered.

Method:
Identify important elements of this MUT. Estimate the required effort to involve all of these in the GI. If there are too many to cover, use information such as the RISK section of the use cases or the judgement of experts to prioritize the elements.

Example:
In a domain model there are static and dynamic models as well as use cases. At least one test case should be created for each use case. There should be sufficient test cases to take every "major" entity through all of its visible states.

Create test cases from the bases

Inputs:
The sampling plan.
MUT

Outputs:
A set of test cases.

Method:
Obtain a scenario from the basis model. Determine the pre-conditions and inputs that are required to place the system in the correct state and to begin the test. Present the scenario to the "oracle" to determine the results expected from the test scenario. Complete a test case description for each test case.

Example:
A different domain expert than the one who supported the model creation would be asked to supply scenarios that correspond to uses of the system. The experts also provide what they would consider an acceptable response.

Apply the checklist and tests to the material

Inputs:
Set of test cases.
Checklist for the type of model being inspected.
MUT

Outputs:
Set of test results.
Completed checklist.

Method:
Apply the test cases to the MUT using the most specific technique available. For UML models in a static environment, such as Rational Rose, an interactive simulation session in which the Creators play the roles of the model elements is the best approach. If the MUT is represented by an executable prototype then the test cases are mapped onto this system and executed. After the model has been thoroughly examined, complete the checklist.

Example:
The domain analysis model is a static UML model. A simulation session is conducted with the Observers feeding test cases to the Creators. The Creators provide details of how the test scenario would be processed through the model. Sequence diagrams are used to document the execution of each test case. Use agreed upon symbols or colors to mark each element that is touched by a test case.

Gather and analyze test results & coverage

Inputs:
Test results in the form of sequence diagrams and pass/fail decisions.
The marked-up model.

Outputs:
Statistics on percentage pass/fail.
Categorization of the results.
Defect catalogs and defect reports.
A judgement of the quality of the MUT and the tests.

Method:

Begin by counting the number of test cases that passed and how many have failed. Compare this ratio to other GIs that have been conducted in the organization. Compute the percentage of each type of element that has been used in executing the test cases. Use the marked-up model as the source of this data. Update the defect inventory with information about the failures from this test session.

Categorize the failed test cases. This can often be combined with the previous two tasks by marking paper copies of the model. Follow the sequence diagram for each failed test case and mark each message, class and attribute touched by a failed test case.

Example:

For the domain analysis model we should be able to report that every use case was the source of at least one test case, that every class in the class diagram was used at least once. Typically on the first pass, some significant states will be missed. This should be noted in the coverage analysis.

Report and feedback

Inputs:

Test results.
Coverage information.

Outputs:

Information on what new tests should be created.
Test report.

Method:

Follow the standard format for a test report in your organization to document the test results and the analyses of those results. If the stated coverage goals are met then the process is complete. If not, use that report to return to step 5 and proceed through the steps to improve the coverage level.

Example:

For the domain analysis tests, some elements were found to be missing from the model. The failing tests might be executed again after the model has been modified.

Roles in the Process

Administrator

The administrative tasks include running the GI sessions, collecting and disseminating the results, and aggregating metrics to measure the quality of the review. In our example, personnel from a central office could do the administrative work.

Creator

The persons who created the MUT are the creators. Depending upon the form that the model takes, these people may "execute" the symbolic model on the test cases or they may assist in translating the test cases into a form that can be executed with whatever representation of the model is available. In our example the modelers who created the domain model would be the "creators".

Observer

Persons in this role create the test cases that are used in the GI. In our example they would be domain experts and preferably experts who were not the source of the information used to create the model initially.

Using Guided Inspection to Validate UML Models

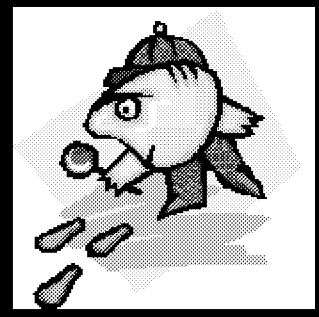
Melissa L. Major

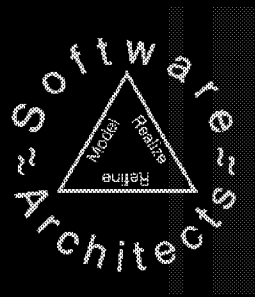
Software Architects

major@software-architects

Problem

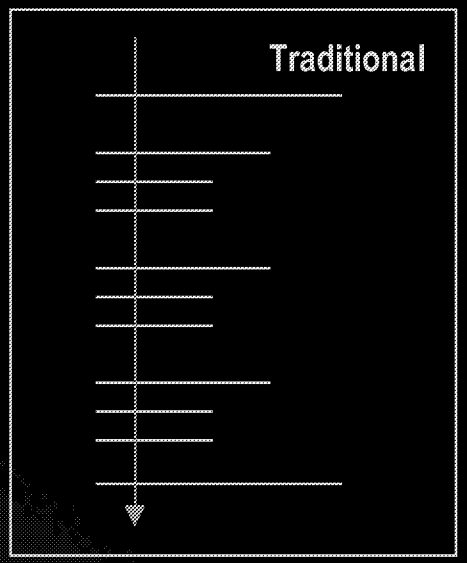
- Existing inspection/review techniques examine ***what is in the model*** for errors.
- There is no systematic way to consider ***what should be in the model***.
- ***Guided Inspection*** is a technique that supplements rigorous inspection/review techniques, that address model syntax, with test cases to systematically examine the semantics of the model.

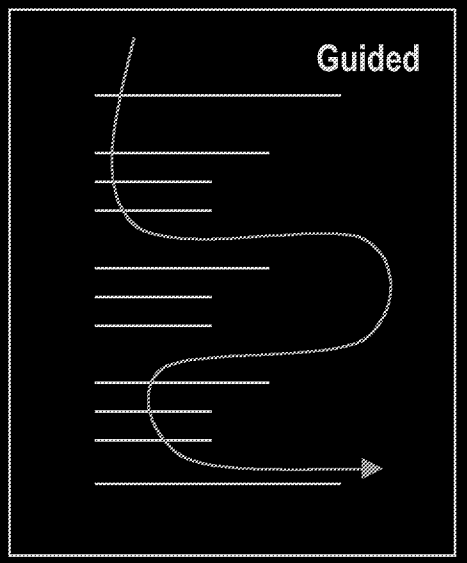




What's Different

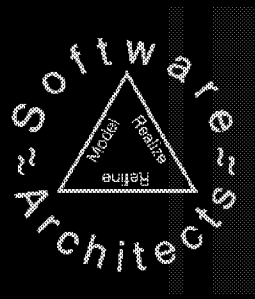
- Guided Inspection does not move sequentially. What is inspected next depends upon the scenario or semantics.
- Inspection can be driven by customer priorities.
- Inspection can be focused to identify specific types of defects.





Guided Inspection Outline

- Analyze the model to be inspected.
- Complete the checklist for the appropriate model.
- Systematically sample to select test cases.
- Write down the test cases.
- Apply the test cases to the model to be inspected.
- Analyze the model to determine coverage levels.



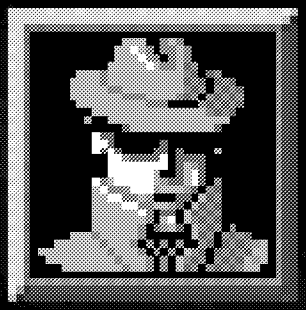
Components in Guided Inspection

- Checklists
  - The tester completes lists by examining the products.
  - The lists are standard across products/projects.
- Test Cases
  - The tester creates test cases.
  - The developer supports a symbolic execution.
  - Tests are specific to the product.

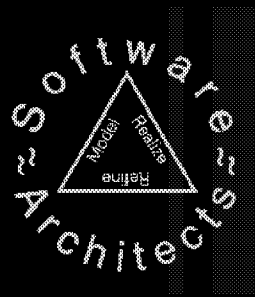
Roles in Guided Inspection

- Tester
  - Select and write test cases.

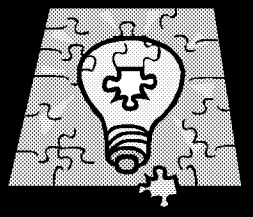
- Developer
  - Perform symbolic execution.



- Manager
  - Stay out of the way - this is defect finding, not a managerial evaluation.

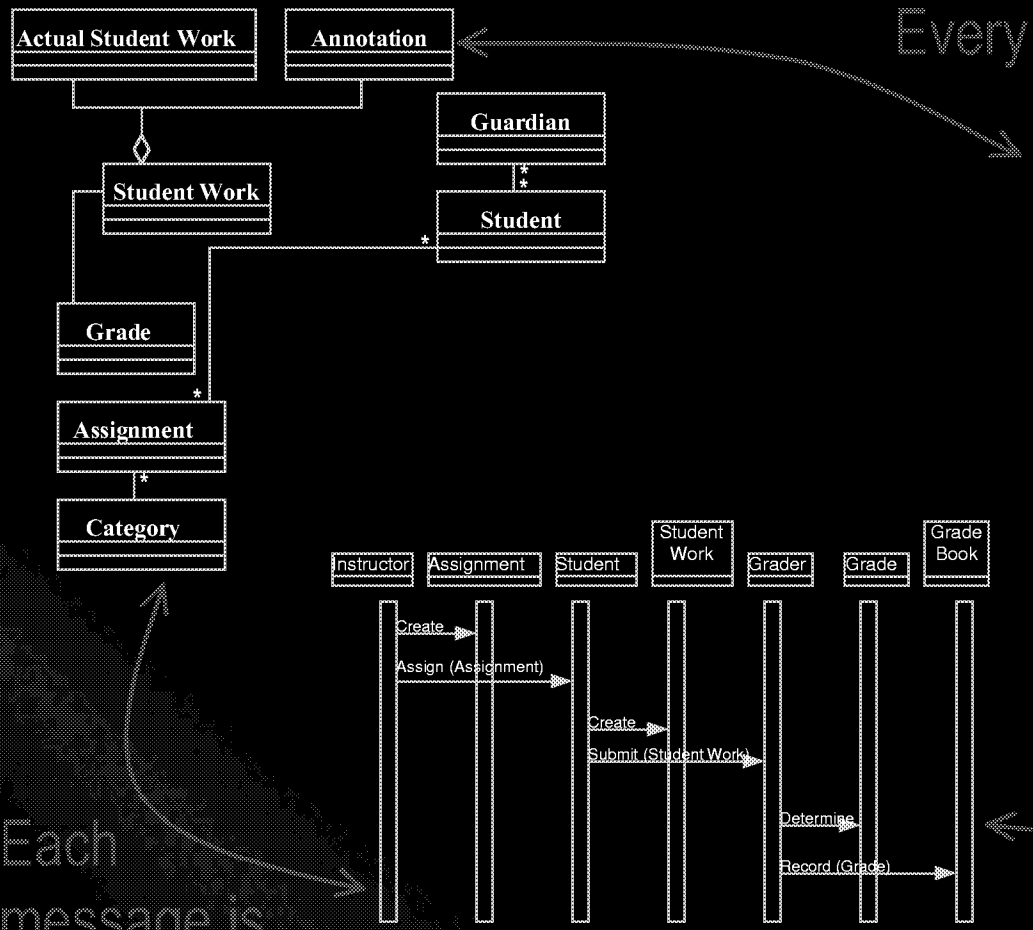


C³ Evaluation Criteria for Models

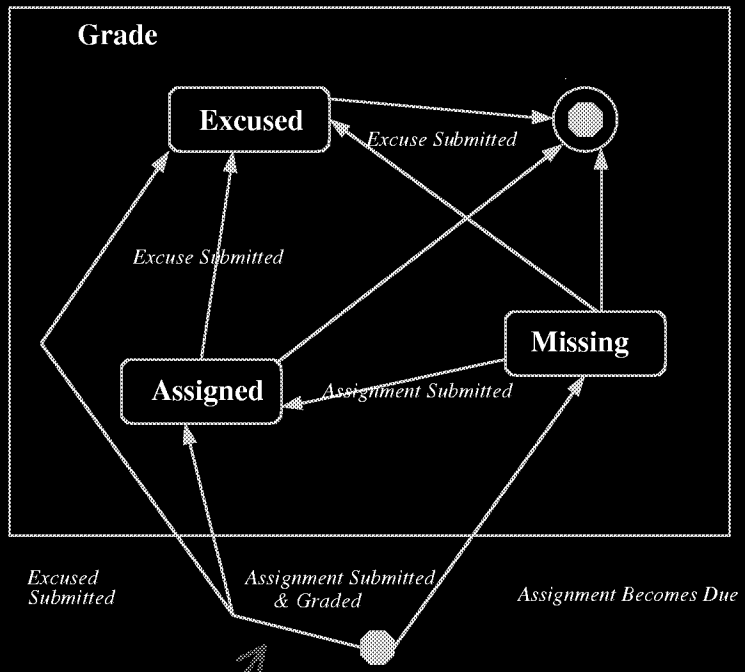


- Completeness
  - Are there scenarios the model can not handle?
  - No required elements are missing.
- Correctness
  - Does the model handle each scenario accurately?
  - Judged equivalent to a reference standard.
- Consistency
  - Are there any contradictions among elements within a work product (internal)?
  - Are there any contradictions between work products (external)?

Consistency Between Diagrams

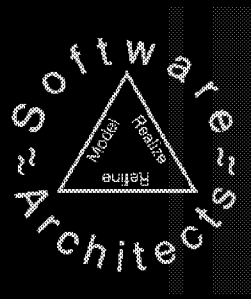


Every trigger is a method



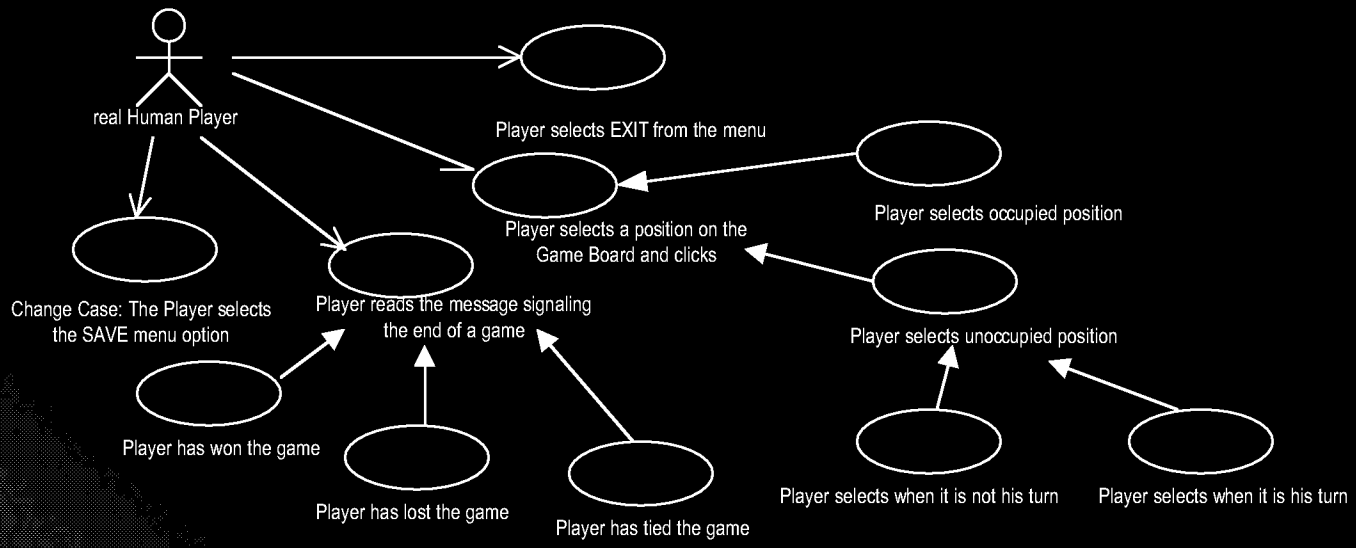
Each message is a method

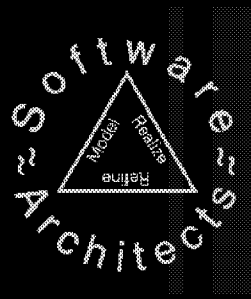
Sequence of messages corresponds to state sequence



Building System Test Cases

- For analysis and high-level design models the test cases can be generated from system use cases.



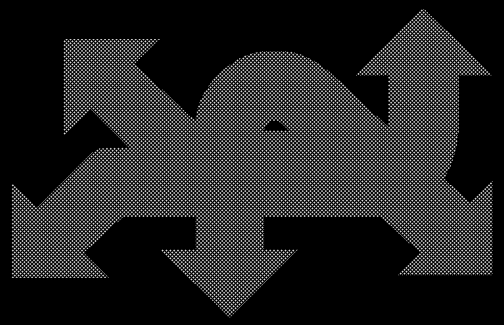


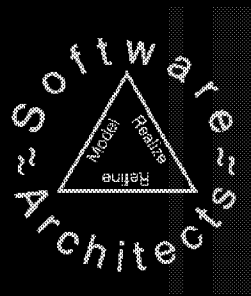
Determining Priorities

- Each use case is annotated with three attributes:
  - Frequency - How often will this feature be used relative to other features of the system?
  - Criticality - How necessary is this feature to the success of the product?
  - Risk - How likely is there to be a problem in implementing this feature?
- Each attribute is valued on a scale from **Low** to **High**.

Combining Attribute Values

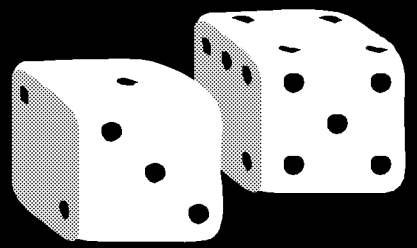
- For a single use case, we have three attribute values.
- Risk is used for scheduling development increments.
- Frequency and criticality are both useful for testing:
  - The most often used, most necessary feature should be tested the most.
- If criticality is **HIGH** and frequency is **LOW:**
  - **Conservative** combined value - **HIGH**
  - **Averaging** combined value - **MEDIUM**

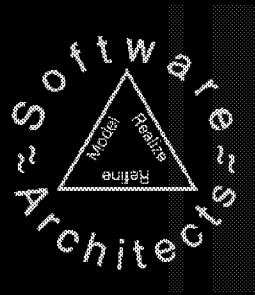




Sampling

- Use cases that have a combined frequency/criticality rating of high will be tested over a wider range than those with a low rating.
- Equivalence classes are established for each variable.
- Test cases are formed by selecting values from the equivalence classes.
- A value for a field is chosen and paired with values of each equivalence class for each variable.



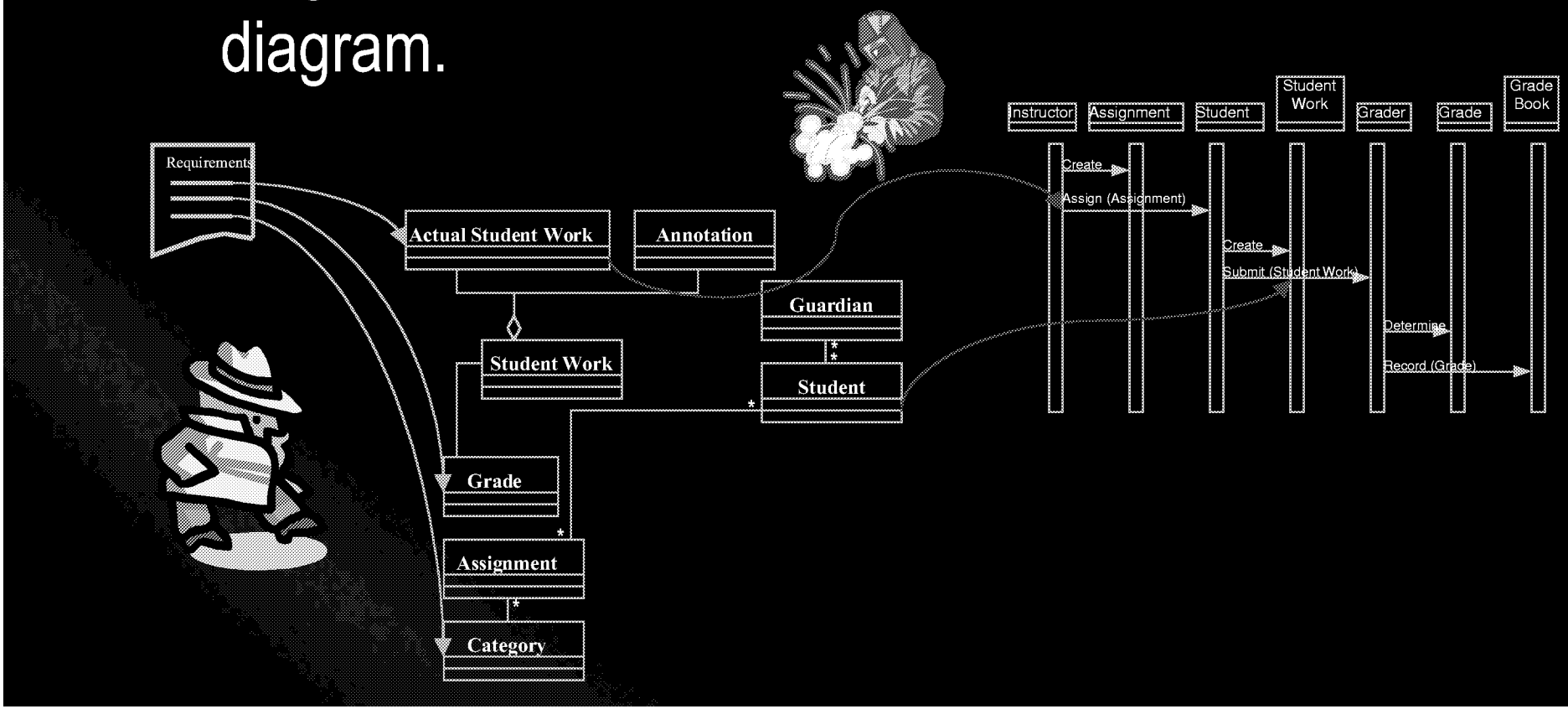


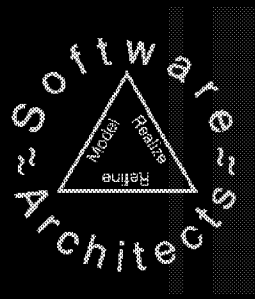
Inspection Session

- Testers guide the inspection by setting the scenario.
- Developers “describe” the execution using their knowledge of the classes, but also referring to pre and post-conditions.
- Developers record the execution using an appropriate UML diagram.

Executing a Test Case

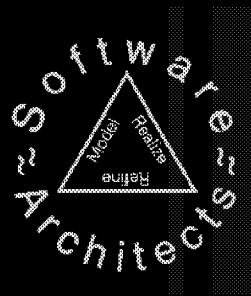
- The scenario guides the inspection of the class diagram. The results are recorded as a sequence diagram.





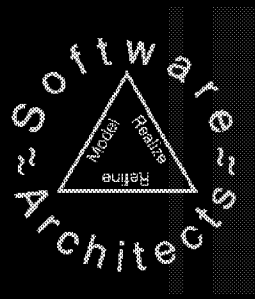
Effectiveness of Guided Inspection

- Data to collect
  - Number of defects detected
  - Number of person hours
- Effectiveness
  - Yield = defects/person hour
- Analysis
  - The bigger the yield the better



Conclusion

- The system has been analyzed from three perspectives: correctness, completeness and consistency.
- Companies have reported that it costs as much as 100 times more to repair a defect at system test time as it would to repair at analysis time.
- While guided inspection is a person intensive technique, even the early expenditure of considerable resources can still result in a net savings over the full project life cycle.



Thanks

- On behalf of Software Architects, thank you for attending this session.
- A more complete presentation of this material is available on our web site:

  www.software-architects.com
- My e-mail address is:

  major@software-architects.com

Please keep in touch if there is anything I can do for you.

Reading Techniques for OO Design Inspections

Guilherme H. Travassos[†,•] travassos@cs.umd.edu
Forrest Shull[‡] fshull@fraunhofer.org
Jeffrey Carver[†] carver@cs.umd.edu
Victor R. Basili[†,‡] basili@cs.umd.edu

[†]**Experimental Software Engineering Group**
Department of Computer Science
University of Maryland at College Park
A.V. Williams Building
College Park, MD 20742
USA

[•]**Computer Science and System Engineering Department COPPE**
Federal University of Rio de Janeiro
C.P. 68511 - Ilha do Fundão
Rio de Janeiro – RJ – 21945-180
Brazil

[‡]**Fraunhofer Center - Maryland**
3115 Ag/Life Sciences Surge Bldg. (#296)
University of Maryland
College Park, MD 20742
USA

ABSTRACT

Inspections can be used to identify defects in software artifacts. In this way, inspection methods help to improve software quality, especially when used early in software development. Inspections of software design may be especially crucial since design defects (problems of correctness and completeness with respect to the requirements, internal consistency, or other quality attributes) can directly affect the quality of, and effort required for, the implementation. We have created a set of "reading techniques" (so called because they help a reviewer to "read" a design artifact for the purpose of finding relevant information) that gives specific and practical guidance for identifying defects in Object-Oriented designs. Each reading technique in the family focuses the reviewer on some aspect of the design, with the goal that an inspection team applying the entire family should achieve a high degree of coverage of the design defects.

In this paper, we present an overview of this new set of reading techniques. We discuss the reading process and how readers can use these techniques to detect defects in high level object oriented design UML diagrams.

Keywords: OO Design, Reading Techniques, Software Quality, and Software Inspection

1. Introduction

A software inspection aims to guarantee that a particular software artifact is complete, consistent, unambiguous, and correct enough to effectively support further system development. For instance, inspections have been used to improve the quality of a system's design and code [Fagan76]. Typically, inspections require individuals to review a particular artifact, then meet as a team to discuss and record defects, which are then sent to the document's author to be corrected. Most publications concerning software inspections have concentrated on improving the inspection meetings while assuming that individual reviewers are able to effectively detect defects in software documents on their own (e.g. [Fagan86, Gilb93]). However, empirical evidence has questioned the importance of team meetings by showing that meetings do not contribute to finding a significant number of new defects that were not already found by individual reviewers [Votta93, Porter95].

"Software reading techniques" attempt to increase the effectiveness of inspections by providing procedural guidelines that can be used by individual reviewers to examine (or "read") a given software artifact and identify defects. These techniques consist of a concrete **procedure** given to a reader on what information in the document to look for. Another important component of the techniques are the **questions** that explicitly ask the reader to think about the information just uncovered in order to find defects. In previous work, we have developed families of reading techniques [Basili96]. There is empirical evidence that software reading is a promising technique for increasing the effectiveness of inspections on different types of software artifacts, not just limited to source code [Porter95, Basili96, Basili96b, Fusaro97, Shull98, Zhang98]. In this work, we concentrate specifically on inspections, for the purpose of defect detection, of high-level Object-Oriented (OO) designs diagrams represented using UML [Fowller97]. (UML is a notational approach that does not define how to organize development tasks.) Figure 1 organizes the "problem space" to which reading techniques can be applied, and illustrates how reading techniques for this task (known as Traceability-Based Reading) fit with previous work. Families of reading techniques have been tailored to defect inspections of requirements (for requirements expressed in English or SCR, a formal notation) and to usability inspections of user interfaces.

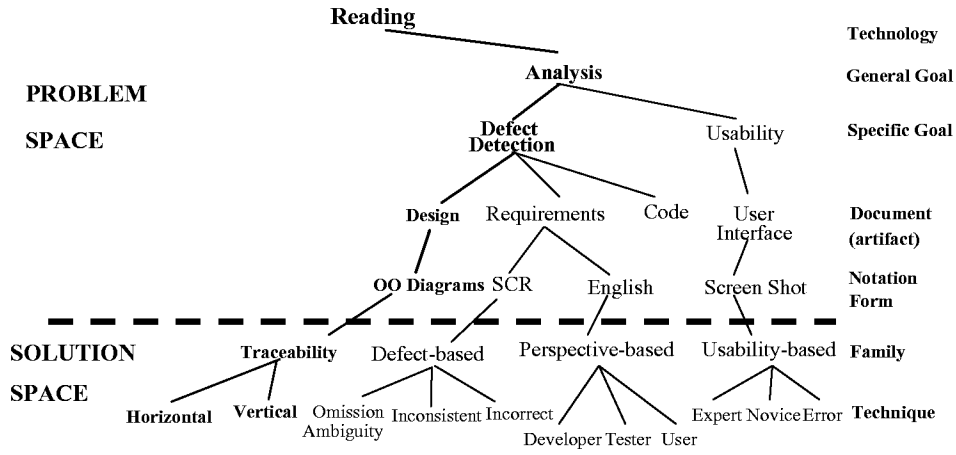


Figure 1 –Families of OO Reading Techniques

Section 2 briefly describes object oriented design in terms of the information that is important to be checked during software inspections. Section 3 introduces the reading techniques, showing the different types of defects such techniques are intended to identify and an outline of the whole set of techniques. The fourth section discusses how the techniques can be used for inspecting OO designs. Finally, some suggestions for future work are discussed in the conclusions.

2. Object Oriented Designs in UML

An OO design is a set of diagrams concerned with the representation of real world concepts as a collection of discrete objects that incorporate both data structure and behavior. Normally, high-level design activities start after the software product requirements are captured. So, concepts must be extracted from the requirements and described using the paradigm constructs. This

means that requirements and design documents are built at different times, using a different viewpoint and abstraction level. When high-level design activities are finished, the documents, basically a set of well-related diagrams, can be inspected to verify whether they are consistent among themselves and if the requirements were correctly and completely captured. High-level design activities deal with the problem description but do not consider the constraints regarding it. That is, these activities are concerned with taking the functional requirements and mapping them to a new notation or form, using the paradigm constructs to represent the system via design diagrams instead of just a textual description. Such an approach allows developers to understand the problem rather than to try to solve it.

Low-level design activities deal with the possible solutions for the problem; they depend on the results from the high-level activities and nonfunctional requirements, and they serve as a model for the code. Our interest is to define reading techniques that could be applied on high-level design documents. We feel that reviews of high-level designs may be especially valuable since they help to ensure that developers have adequately understood the problem before defining the solution. Since low-level designs use the same basic diagram set as the high-level design, but using more detail, reviews of this kind can help ensure that low-level design starts from a high-quality base.

More specifically, the reading techniques investigated in this work are tailored to inspections of documents using UML notation. UML diagrams capture the static and dynamic view of the real world as described by the object-oriented constructs. We focused our reading techniques on the following high-level design diagrams: class, interaction (sequence and collaboration), state machine and package. Usually, these are the main UML diagrams that developers build for high-level OO design. They capture the static and dynamic views of the problem, and even allow the teamwork to be organized, based on packaging information. The design content needs to be compared against the requirements, which can likewise be described using a number of separate diagrams to capture different aspects. In particular, we expect that there will be a textual description of the functional requirements that may also describe certain behaviors using more specialized representations such as use-cases [Jacobson95].

Thus, we identify the following as important sources of information for ensuring the quality of a UML high level design:

- A set of functional requirements that describes the concepts and services that are necessary in the final system;
- Use cases that describe important concepts of the system (which may eventually be represented as objects, classes, or attributes) and the services it provides;
- A class diagram (possibly divided into packages) that describes the classes of a system and how they are associated;
- A set of class descriptions that lists the classes of a system along with their attributes and behaviors;
- Sequence diagrams that describe the classes, objects, and possibly actors of a system and how they collaborate to capture services of the system;
- State diagrams that describe the internal states in which a particular object may exist, and the possible transitions between those states.

3. Reading Techniques for high-level design

Each reading technique can be thought of as a set of procedural guidelines that reviewers can follow, step-by-step, to examine a set of diagrams and detect defects. The types of defects on which our techniques are focused, as listed in Table 1, are based on earlier work with requirements inspections. The defect taxonomy is important since it helps focus the kinds of questions reviewers should answer during an inspection.

Type of Defect	***Description***
Omission	One or more design diagrams that should contain some concept from the general requirements or from the requirements document do not contain a representation for that concept.
Incorrect Fact	A design diagram contains a misrepresentation of a concept described in the general requirements or requirements document.
Inconsistency	A representation of a concept in one design diagram disagrees with a representation of the same concept in either the same or another design diagram.
Ambiguity	A representation of a concept in the design is unclear, and could cause a user of the document (developer, low-level designer, etc.) to misinterpret or misunderstand the meaning of the concept.
Extraneous Information	The design includes information that, while perhaps true, does not apply to this domain and should not be included in the design.

Table 1 – Types of software defects, and their specific definitions for OO designs

We defined one reading technique for each pair or group of diagrams that could usefully be compared against each other. For example, use cases needed to be compared to interaction diagrams to detect whether the functionality described by the use case was captured and all the concepts and expected behaviors regarding this functionality were represented. The full set of our reading techniques is defined as illustrated in Figure 2, which differentiates horizontal[1] (comparisons of documents within a single lifecycle phase) from vertical[2] (comparisons of documents between phases) reading.

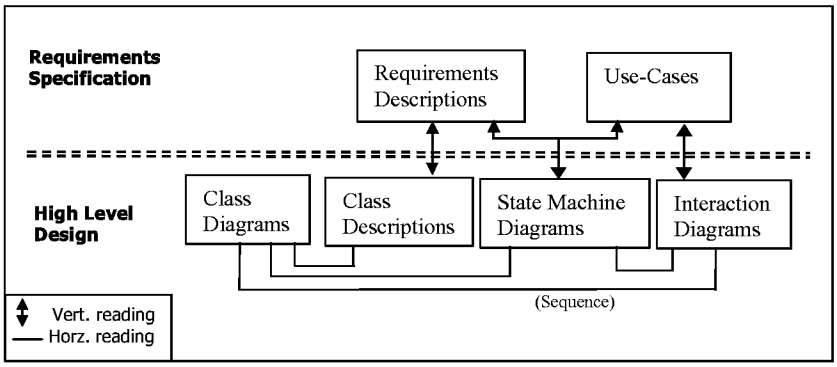


Figure 2 – Set of OO Reading Techniques

[1] Consistency among documents is the most important feature here.
[2] Traceability between the phases is the most important feature here.

Initial validation of these techniques was accomplished by means of a study [Shull99, Travassos99] that provided evidence for the feasibility of these techniques. Using the techniques did allow teams to detect defects, and in general subjects agreed that the techniques were helpful. Also, the vertical techniques tended to find more defects of omitted and incorrect functionality, while the horizontal techniques tended to find more defects of ambiguities and inconsistencies between design documents, lending some credence to the idea that the distinction between horizontal and vertical techniques is real and useful [Travassos99].

Further studies have been undertaken to improve the practical applicability of the techniques. As a result of specific feedback from the feasibility study, we developed a second version of the techniques and studied them using an observational approach (i.e., using experimental methods suitable for understanding the process by which subjects apply the techniques) [Travassos99b]. The feasibility study had identified *global* issues for improvement, that is, issues that affected the entire process, such as the amount of semantic versus syntactic checking. The observational approach was necessary to understand what improvements might be necessary at the level of individual steps, for example, whether subjects experience difficulties or misunderstandings while applying the technique (and how these problems may be corrected), whether each step of the technique contributes to achieving the overall goal, and whether the steps of the technique should be reordered to better correspond to subjects' own working styles. Detailed information about the results and also an improved version of the techniques can be found in [Shull99b].

4. Using OO Reading Techniques for inspecting OO Design

In this section we explore the application of the reading techniques in an inspection process. While horizontal reading aims to identify whether all of the design artifacts are describing the same system, vertical reading tries to verify whether those design artifacts represent the right system, which is described by the requirements and use-cases. So, the goal is that when all the techniques are used together, then all the quality issues in the design are covered. The development team can use the whole set of the techniques, but if some design artifacts do not exist, there is no impact on the design inspection process. A subset or reordering of the techniques may also be chosen based on important attributes of the design to be reviewed. This is particularly interesting when developers are dealing with specialized application domains. For example, consider a system whose functionality is based mainly on its reaction to stimuli where state machine diagrams are common. In this situation, it could be beneficial to use the reading techniques that focus on state machine diagrams before using the reading techniques that focus on the other design diagrams. For conventional systems, such as database systems, the semantic model of the information and the flow of the transactions seem to be the important information. Therefore, a subset of the techniques could be picked that focus on this information. In this situation, first reading the class diagram against the sequence diagrams seems to be a good idea then continuing with the rest of the techniques.

To organize the reading process, reading responsibilities can de distributed among the members of the inspection team, reducing the reading effort per team member and improving the reading process. In this way, each one of the readers can apply a reduced number of reading techniques, or even deal with a reduced number of artifacts at the same time. After individual review, it is important to organize a meeting in order to review each one of the individual defect lists and to create a final list that reflected a group consensus of the defects in the documents. It is not necessary to apply the techniques in a particular order, but it seems to be reasonable to apply first horizontal reading for all existing design artifacts and then vertical reading, to ensure that a consistent system description is checked against the requirements. In Figure 3 is an example of how the techniques could be organized among a team of three reviewers.

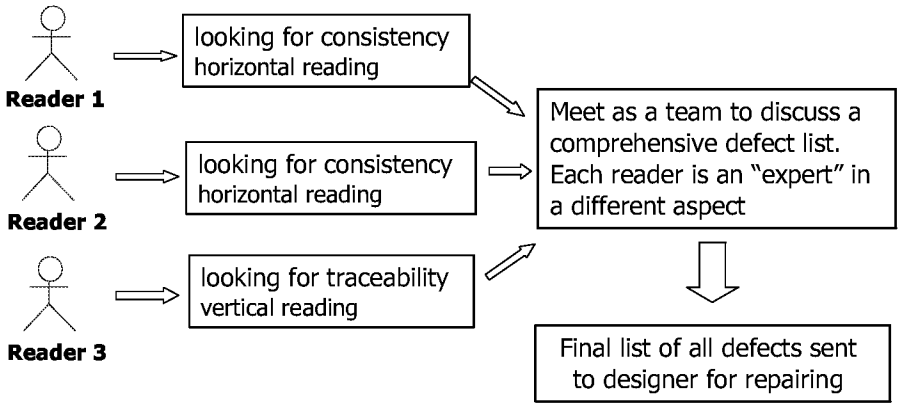


Figure 3 – Organizing reading with 3 readers

To support these two types of reading (horizontal and vertical) we have introduced some new terminology to describe the actions of the system. First, because the level of abstraction and granularity of the information in the requirements and use-cases is different from the abstraction and information in the design artifacts, the concept of *system functionality* was broken down into three complementary concepts (messages, services, and functionality). Messages are the very lowest-level behaviors out of which system services and, in turn, functionalities are composed. They represent the communication between objects that work together to implement system behavior. Messages may be shown on sequence diagrams and must be associated with class behaviors. Services are combinations of one or more messages and usually capture some basic activity necessary to accomplish a functionality. They can be considered low-level actions performed by the system. They are the "atomic units" out of which system functionalities are composed. A service could be used as a part of one or more functionalities. We use the term "functionality" to describe the behavior of the system from the user's point of view, in other words, the functionality that the user expects to be visible. A functionality is composed of one or more services. Users do not typically consider services an end in themselves; rather, services are the steps by which some larger goal or functionality is achieved.

A second important piece of terminology is that of conditions and constraints. A condition describes what must be true for the functionality to be executed. A constraint must always be true for system functionality. This information is important to readers comparing different diagrams

since it describes *how* the functionality must be implemented; this information is important to maintain with the functionality it describes.

B.2 Reading 2 -- State diagrams x Class description

Goal: To verify that the classes are defined in a way that can capture the functionality specified by the state diagrams.

Inputs to Process: A set of class descriptions that lists the classes of a system along with their attributes and behaviors and a state diagram that describes the internal states in which an object may exist, and the possible transitions between states.

For **each state diagram,** perform the following steps:

1) **Read the state diagram to understand the possible states of the object and the actions that trigger transitions between them.**
2) **Find the class or class hierarchy, attributes, and behaviors on the class description that correspond to the concepts on the state diagram.**
3) **Compare the class diagram to the state diagram to make sure that the class, as described, can capture the appropriate functionality.**

   Using your semantic knowledge of this class and the behaviors it should encapsulate, are all states described? If not, you have uncovered a defect of incorrect fact, that is, the class as described cannot behave as it should.

   Is there some unstarred state? Could you evaluate the importance of this state? Does it really describe an essential object state? Is the state feasible considering all actions and constraints surrounding it? If yes, probably something is missing on the class diagram and there is an inconsistency between the diagrams. Otherwise, an extraneous fact should be reported.

   Is there some unstarred event? If yes, fill in a defect record showing the inconsistency between the class description and state diagram.

   Is there some unstarred constraint? Is the constraint directly concerned with some object data? If yes, fill in a defect record showing the information that has been omitted from the class description.

Figure 4 – An excerpt of a Horizontal Reading

The main idea in applying horizontal reading is to understand whether all the high level design artifacts are representing the same system. We must keep in mind that the artifacts should model the same system information but from different perspectives. UML organizes the artifacts and different types of information based on the type of system information they contain. There are specific artifacts to capture essentially static information (basically, the structure assumed by the domain's objects while playing specific roles in the problem domain) and specific artifacts to capture essentially dynamic information (basically, the consequences when objects are asked to behave in order to accomplish system functionalities). These different views are useful and together allow developers to understand what is going on with the objects and how they are accomplishing the required functionalities in the context of the problem. However, these differences among the diagrams make the inspection process a bit more complicated. For instance, when comparing sequence diagrams against state machine diagrams two different perspectives must be combined to interpret and identify possible defects. Each one of the sequence diagrams is a represents some system objects and the messages exchanged between them that implement some functionality required by the user while, on the other hand, the state machine diagram is a picture of what happens to one object when it is influenced by the events occurring in multiple sequence diagrams. Sequence diagrams show the specific messages exchanged by objects, while state diagrams show how the system responds to events, which can be messages, services, or functionality. Both diagrams must convey information about conditions

and constraints on the functionality. So, the horizontal reading techniques explore these types of differences and help reduce the semantic gap between the documents. Figure 4 shows an excerpt from a horizontal reading technique highlighting the concerns for each one of the reading steps (some details are omitted).

B.7 Reading 7 -- State Diagrams x Requirements Description and Use-cases
Goal: To verify that the state diagrams describe appropriate states of objects and events that trigger state changes as described by the requirements and use cases.
Inputs to process: The set of all state diagrams, each of which describes an object in the system. A set of functional requirements that describes the concepts and services that are necessary in the final system and the set of use cases that describe the important concepts of the system

For each state diagram, do the following steps:

1) **Read the state diagram to basically understand the object it is modeling.**
2) **Read the requirements description to determine the possible states of the object, which states are adjacent to each other, and events that cause the state changes.**
3) **Read the Use cases and determine the events that can cause state changes.**
4) **Read the state diagram to determine if the states described are consistent with the requirements and if the transitions are consistent with the requirements and use cases.**

Were you able to find all of the states?

If a state is missing, look to see if two or more states that you marked in the requirements were combined into one state on the state diagram. If not, then you have found a defect of Omission. If so, then does this combination make sense? If not, you have found a defect of Incorrect Fact.

Were there extra states in the state diagram?

Look to see if one state that you marked in the requirements has been split into two or more states in the state diagram. If not, then you have found a defect of Extraneous. If so, does this split make sense? If not, you have found a defect of Incorrect Fact.

Do all of the events on the adjacency matrix appear on the state diagram? If not, you have found a defect of omission. Do events appear on the state diagram that are not on the adjacency matrix? If so, you have found a defect of extraneous fact.

Did you find all of the constraints that are on the adjacency matrix? If not, then you have found a defect of omission. Did you find a constraint on the state diagram that is not on the adjacency matrix? If so, does the constraint make sense? If not then you have found a defect of extraneous fact.

Figure 5 – An excerpt of a Vertical Reading

To apply vertical reading readers should be aware of the differences between the two lifecycle phases in which the documents were created and how the traceability between these two different phases could be explored. The levels of abstraction and information representation between these phases are quite different. Requirements and use cases should precisely describe the problem and thus use a totally different representation than the design artifacts. Moreover, usually the entire problem definition is presented using these two types of document. There is no separation of concerns and no direct mapping from one phase (specification) to another (design). Vertical reading techniques explore such ideas and provide some guidance to help the reader identify the information s/he needs. For example, the requirements descriptions and use cases capture the functionality of the entire system and in some cases the services, but not the messages. Designers using these requirements and use cases decide about the messages based on the viewpoint (abstraction) used to classify and organize the classes. Sequence diagrams are organized based on messages that work together in some way to provide the services, which compose the required functionality. Requirements and use cases describe constraints and conditions in general terms;

on a sequence diagram such information must be made explicit and associated with the appropriate messages. So, vertical reading techniques explore these types of differences by defining some guidelines for tracing the right information between these two lifecycle phases. Figure 5 shows an excerpt from a vertical reading technique highlighting the concerns for each one of the reading steps (some details are omitted).

A full description of the entire set of techniques, including the ones referred to here, can be found in [Shull99b], which is accessible via the web.

5. Ongoing Work

The Object Oriented reading techniques (OORTs) have been, and still are, evolving since their first definition. New issues and improvements have been included based on the feedback of readers and volunteers. Throughout this process, we have been trying to capture new features and to understand whether the latest version of the reading techniques keeps its feasibility and interest. We have found observational techniques useful, because they have allowed us to follow the reading process as it occurred, rather than trying to interpret the readers' post-hoc answers as we have done in the past. Observing how readers normally try to read diagrams challenged many of our assumptions about how our techniques were actually being applied.

However, two important questions remain open in this area. First, the role of domain knowledge is not yet well understood for these two sets of reading techniques, especially for horizontal reading. Since horizontal reading is a largely syntactic check of consistency between two design diagrams, it is not expected to require domain knowledge. Still, it has been observed that a reader possessing some knowledge about the problem domain seemed to be more effective than a reader who does not have the same level of knowledge. Some empirical investigation into exactly how domain knowledge plays a role in this type of reading could help us better understand and thus better support the process. The second question regards the level of automated support that should be provided for such techniques. The observational studies have allowed us to understand which steps of the techniques can feel especially repetitive and mechanical to the reader. So, the clerical activities regarding the reading process using OORTs must be precisely defined and identified. For this situation, further observational studies play an important role and they should be executed aiming to collect suggestions on how to automate the clerical activities concerned with OORTs.

Currently, the techniques are undergoing experimental evaluation, which is aimed at evolving them. In each experiment we explore a different issue regarding the techniques in order to evolve them or understand them at a deeper level. This series of experiments is an evolutionary process. The feedback from the readers and the observation of the techniques usage are playing an important role as we work towards a useful and feasible set of reading techniques for OO design. The results of these experiments will be published in future publications, which will be available at http://www.cs.umd.edu/projects/SoftEng/ESEG.

Acknowledgements

This work was partially supported by UMIACS and by NSF grant CCR9706151.
Dr. Travassos also recognizes the partial support from CAPES- Brazil.

References

[Basili96] V. R. Basili, S. Green, O. Laitenberger, F. Lanubile, F. Shull, S. Sorumgard, M. V. Zelkowitz. The Empirical Investigation of Perspective-Based Reading, Empirical Software Engineering Journal, I, 133-164, 1996.

[Basili96b] V. Basili, G. Caldiera, F. Lanubile, and F. Shull. Studies on reading techniques. *In Proc. of the Twenty-First Annual Software Engineering Workshop*, SEL-96-002, pages 59-65, Greenbelt, MD, December 1996.

[Fagan76] M. E Fagan. "Design and Code Inspections to Reduce Errors in Program Development." IBM Systems Journal, 15(3):182-211, 1976.

[Fagan86] M. Fagan. "Advances in Software Inspections." IEEE Transactions on Software Engineering, 12(7): 744-751, July 1986.

[Fowller97] M. Fowller, K. Scott. UML Distilled: Applying the Standard Object Modeling Language, Addison-Wesley, 1997.

[Fusaro97] P. Fusaro, F. Lanubile, and G. Visaggio. A replicated experiment to assess requirements inspections techniques, *Empirical Software Engineering Journal*, vol.2, no.1, pp.39-57, 1997.

[Gilb93] T. Gilb, D. Graham. Software Inspection. Addison-Wesley, reading,MA, 1993.

[Jacobson95] I. Jacobson, M. Christerson, P. Jonsson, G. Overgaard. Object-Oriented Software Engineering: A Use Case Driven Approach, Addison-Wesley, revised printing, 1995.

[Porter95] A. Porter, L. Votta Jr., V. Basili. Comparing Detection Methods for Software Requirements Inspections: A Replicated Experiment. IEEE Transactions on Software Engineering, 21(6): 563-575, June 1995.

[Shull98] F. Shull. *Developing Techniques for Using Software Documents: A Series of Empirical Studies*. Ph.D. thesis, University of Maryland, College Park, December 1998.

[Shull99] F. Shull, G. Travassos, V. Basili. Towards Techniques for Improved OO Design Inspections. Workshop on Quantitative Approaches in Object-Oriented Software Engineering (in association with the 13th European Conf. on Object-Oriented Programming), Lisbon, Portugal, 1999. On line at http://www.cs.umd.edu/projects/SoftEng/ESEG/papers/postscript/ecoop99.ps.

[Shull99b] Forrest Shull, Guilherme H. Travassos, Jeffrey Carver, Victor R. Basili. Evolving a Set of Techniques for OO Inspections. Technical Report CS-TR-4070, UMIACS-TR-99-63, University of Maryland, October 1999. http://www.cs.umd.edu/Dienst/UI/2.0/Describe/ncstrl.umcp/CS-TR-4070

[Travassos99] G. Travassos, F. Shull, M. Fredericks, V. Basili. Detecting Defects in Object-Oriented Designs: Using Reading Techniques to Improve Software Quality. In the Proceedings of the Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA), Denver, Colorado, 1999.

[Travassos99b] Guilherme H. Travassos, Forrest Shull, Jeffrey Carver. Evolving a Process for Inspecting OO Designs. XIII Brazilian Symposium on Software Engineering: *Workshop on Software Quality*. Florianópolis, Curitiba, Brazil, October 1999. On line at http://www.cs.umd.edu/projects/SoftEng/ESEG/papers/ postscript/wqs99.ps.

[Votta93] L. G.Votta Jr. "Does Every Inspection Need a Meeting?" ACM SIGSOFT Software Engineering Notes, 18(5): 107-114, December 1993.

[Zhang98] Z. Zhang, V. Basili, and B. Shneiderman. An empirical study of perspective-based usability inspection. Human Factors and Ergonomics Society Annual Meeting, Chicago, Oct. 1998.

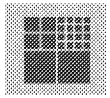
Reading Techniques for OO Design Inspections

Guilherme H. Travassos
Forrest Shull
Jeff Carver
Victor R. Basili

{travassos, basili, carver}@cs.umd.edu
fshull@fraunhofer.org

http://www.cs.umd.edu/projects/SoftEng/ESEG/

UNIVERSITY OF MARYLAND
Department of Computer Science
Experimental Software Engineering Group



Fraunhofer Center - Maryland

Reading Techniques

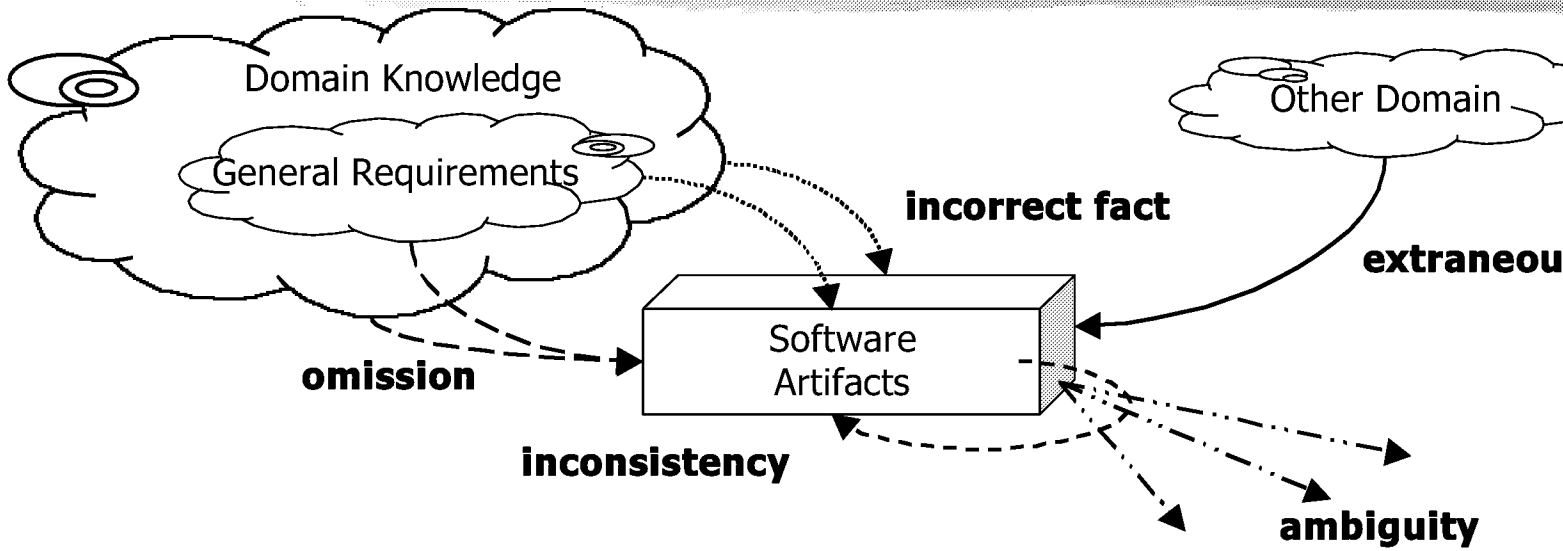
Why read?

Software practitioners are taught how to write, but typically not how to read, software documents

Many software processes assume that practitioners know how to effectively find the information they need (i.e. how to read) such documents

e.g. inspection process: read a document to find defects

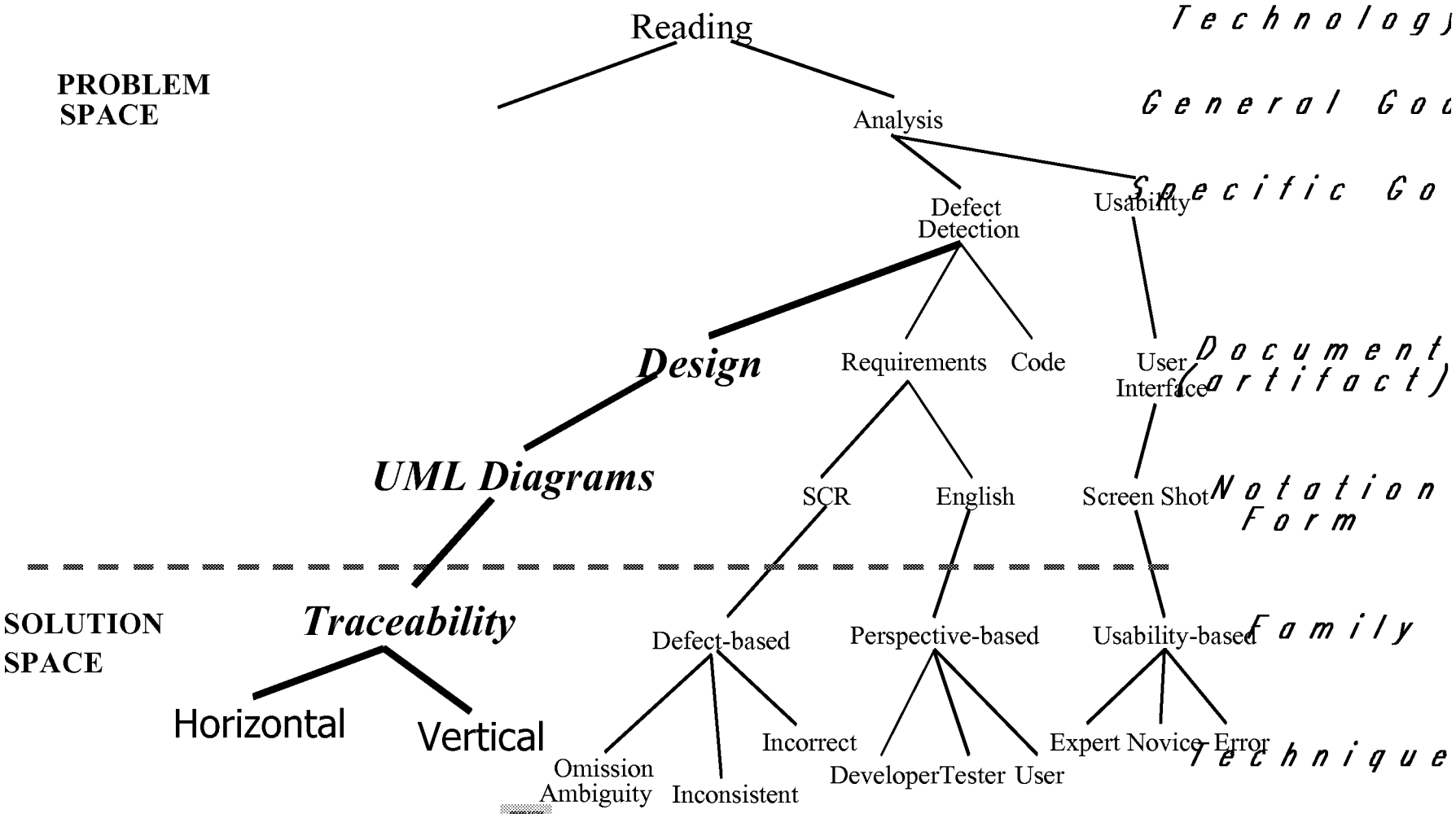
Reading Techniques



- "Software reading techniques" attempt to increase the effectiveness of inspections by providing procedural guidelines that can be used by individual reviewers to examine (or "read") a given software artifact and identify defects

- There is empirical evidence that software reading is a promising technique for increasing the effectiveness of inspections on different types of software artifacts, not just limited to source code.

Reading Techniques for OO Design



Reading Techniques for OO Design

looking for defects:

Type of Defect	*Description*
Omission	One or more design diagrams that should contain some concept from the general requirements or from the requirements document do not contain a representation for that concept.
Incorrect Fact	A design diagram contains a misrepresentation of a concept described in the general requirements or requirements document.
Inconsistency	A representation of a concept in one design diagram disagrees with a representation of the same concept in either the same or another design diagram.
Ambiguity	A representation of a concept in the design is unclear, and could cause a user of the document (developer, low-level designer, etc.) to misinterpret or misunderstand the meaning of the concept.
Extraneous Information	The design includes information that, while perhaps true, does not apply to this domain and should not be included in the design.

Table 1 – Types of software defects, and their specific definitions for OO designs

OO Software Design Process with

Unified Modeling Language - UML

- **UML is just a notational approach and does not propose/define how to organize the design tasks.**
- **Therefore, it can be tailored to fit different development situations and software life-cycles**

UML Design Artifacts

Dynamic View

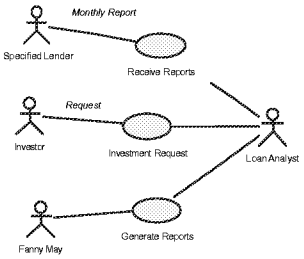
- *use cases*
- activities
- interaction
  - sequences
  - collaborations
- state machines

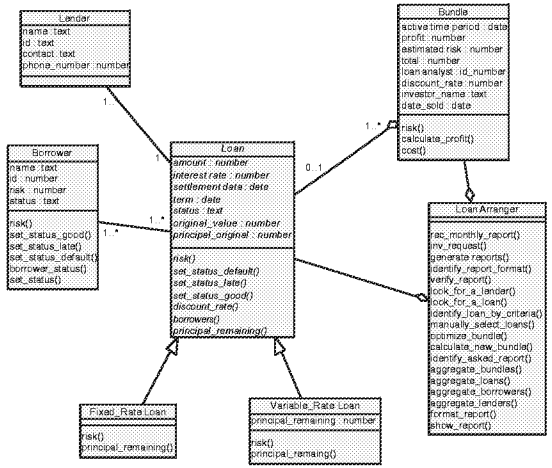
Static View

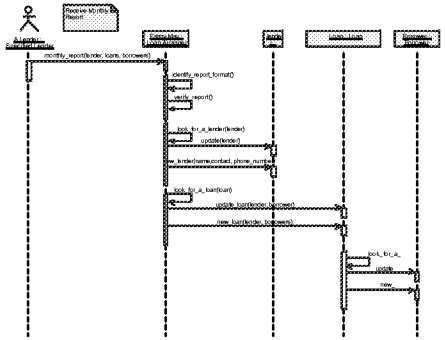
- classes
  - Relationships
    - Association
    - Generalization
    - Dependency
    - Realization
  - Extensibility
    - Constraints
    - Tagged values
    - Stereotypes
- packages
- deployment

Reading Techniques for OO Design

UML Artifacts:







Loan-Arranger Requirements Specification – Jan. 8, 1999

Background

Banks generate income in many ways, often by borrowing money from their depositors at a low interest rate, and then lending that same money at a higher interest rate in the form of bank loans. However, property loans, such as mortgages, typically have terms of 15, 25 or even 30 years. For example, suppose that you purchase a $150,000 house with a $50,000 down payment and borrow a $100,000 mortgage from National Bank for thirty years at 5% interest. That means that National Bank gives you $100,000 to pay the balance on your house, and you pay National Bank back at a rate of 5% per year over a period of thirty years. You must pay back both principal and interest. That is, the initial principal, $100,000, is paid back in 360 installments (once a month for 30 years), with interest on the unpaid balance. In this case the monthly payment is $536.82. Although the income from interest on these loans is lucrative, the loans tie up money for a long time, preventing the banks from using their money for other transactions. Consequently, the banks often sell their loans to consolidating organizations such as Fannie Mae and Freddie Mac, taking less long-term profit in exchange for freeing the capital for use in other ways.

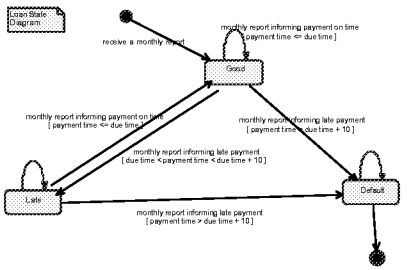
Loan Arranger Classes Description

Class name: Fixed_Rate Loan
Category: Logical View
Documentation:
A fixed rate loan has the same interest rate over the entire term of the mortgage

External Documents:
Export Control: Public
Cardinality: n
Hierarchy:
Superclasses: Loan
Public Interface:
Operations:
risk
principal_remaining

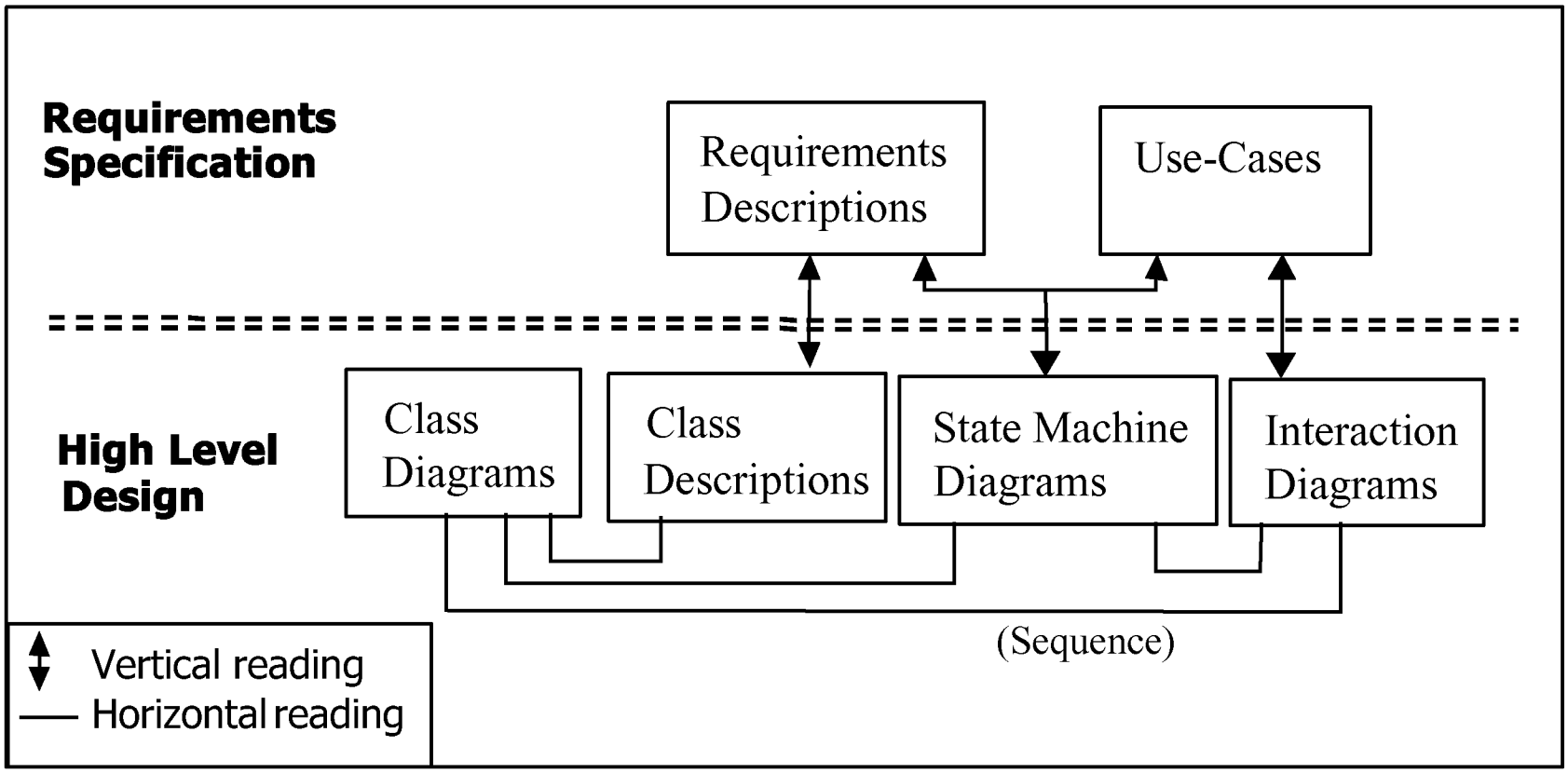
State machine: No
Concurrency: Sequential
Persistence: Persistent

Operation name: risk
Public member of: Fixed_Rate Loan
Return Class: float
Documentation:
take the average of the risks' sum of all borrowers related to this loan
if the average risk is less than 1 round up to 1
else if the average risk is less than 100 round up to the nearest integer
otherwise round down to 100
Concurrency: Sequential



Reading Techniques for OO Design

Target Artifacts:



Reading Techniques for OO Design

Reading Technique for Class diagrams x Class descriptions

...

2) Read the **class descriptions**

...

Look at all the attributes are described along with basic types

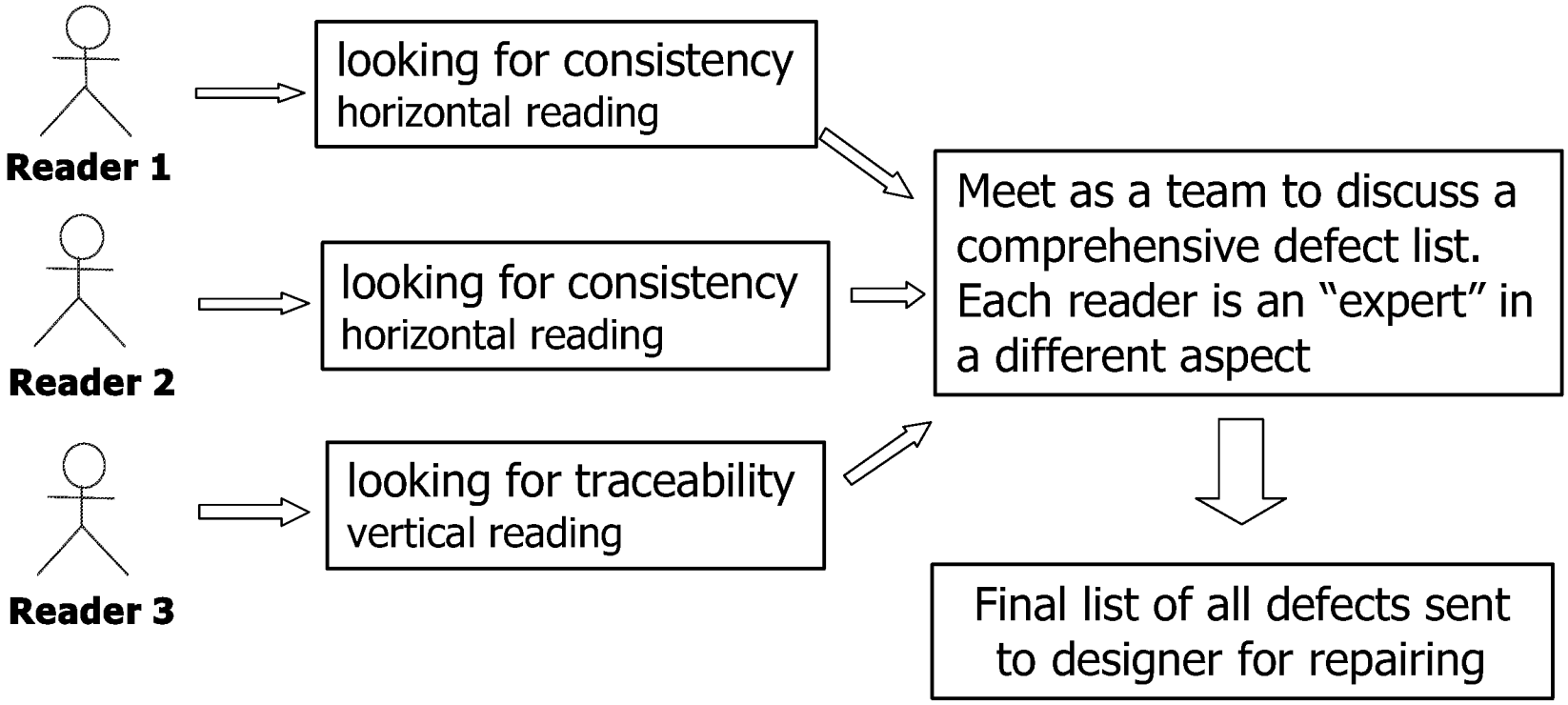
Can this class encapsulate all these attributes? Are the attributes associated to feasible/possible basic types? Does it make sense to have these attributes in the class description? Is some attribute missing between the two documents or sounding extraneous?

...

3) …

Reading Techniques for OO Design Team Organization

The inspection process with OO reading techniques:



Horizontal Reading Techniques

- **Ensure all design artifacts represent the same system**
- **Design contain complementary views of the information**
  - Static (class diagrams)
  - Dynamic (interaction diagrams)
- **Not obvious how to compare these different perspectives**

Vertical Reading Techniques

- **Ensure that the design artifacts represent the same system described by the requirements and use-cases**
- **Comparing documents from different lifecycle phases**
  - Level of abstraction and detail are different

How we are learning

Empirical Evaluations

- Receiving feedback from users of the techniques
  - Controlled Experiments
  - Observational Studies
- Modifying the techniques based on feedback
  - Qualitative
  - Quantitative
- Continually evaluating the techniques to ensure they remain feasible and useful

How we are learning

- **Controlled Experiment**
  - Undergraduate Software Engineering class
  - Goal: Feasibility and Global Improvement
- **Observational Studies**
  - Goal: Feasibility and Local Improvements
- **Controlled Experiment II**
  - Graduate Software Engineering Class
  - Goal: Observation and Local Improvement

What we know so far

- **Techniques are feasible**
- **Techniques help find defects**
- **Vertical reading finds more defects of omission and incorrect fact**
- **Horizontal reading finds more defects of inconsistency and ambiguity**
- For more details:
  http://www.cs.umd.edu/Dienst/UI/2.0/Describe/ncstrl.umcp/CS-TR-4070

What we don't know

- **What influence does domain knowledge have on the reading process?**
  - Horizontal X Vertical
- **Can we automate a portion of the techniques?**
  - Some steps are repetitive and mechanical
  - Need to identify clerical activities

Session 3: Software Process Improvement

Christian Halvorsen, Norwegian University of Science and Technology

Stan Rifkin, Master Systems

Nancy Eickelman, NASA/IV&V

A Taxonomy of SPI Frameworks

Christian Printzell Halvorsen, Reidar Conradi
Norwegian University of Science and Technology (NTNU)
N-7491 Trondheim, Norway
Phone: +47 73 59 34 44 Fax: +47 73 59 44 66
{cph, conradi}@idi.ntnu.no

Abstract

There exist a number of different approaches, often called *frameworks,* supporting *software process improvement (SPI).* Their differences and similarities has been the subject of some debate. This paper discusses four different classes of methods, which can be used to compare SPI frameworks. One of these methods is a new taxonomy proposed in this paper.

1. Introduction

Focus on *software process improvement (SPI)* is growing. The underlying assumption of SPI is that product quality is influenced by the quality of the process used to produce it:

$$\textbf{Quality}(\textit{Process}) \Rightarrow \textbf{Quality}(\textit{Product})$$

This causal relation may seem trivial at first, but in reality there are numerous variations in the approach to SPI. These approaches are often called *SPI frameworks* and they generally describe how organizations can assess current process quality, as well as how they can improve it. Most frameworks are rather comprehensive and differences in content are evident in a number of aspects, e.g. focus, goals, adaptability and so on. There are even subtle differences in their interpretation of words like *quality* and *process.*

However, the SPI framework differences may not be apparent at first, and because the frameworks are so comprehensive, it is costly to investigate them all. The result is that the differences, which set one framework apart from another, are not clear. Evidently, systematic methods to compare the frameworks are needed. The question is how this can be done *efficiently, objectively* and in a way that is *possible to validate.*

1.1 Why Compare SPI Frameworks?

Comparing SPI frameworks can be rewarding from an academic view. However, focus should not be on the frameworks themselves, but on real improvements resulting from their adoption. SPI framework comparisons should therefore provide practical insight and guidance when selecting which framework to employ in a software-producing organization. It should be clear that no single "right" comparison method exists for this purpose, and a combination of methods may be necessary depending on the context. The primary usability requirements to be considered are:

- *Knowledge-level* – The amount of detail in the comparison should correspond to the knowledge-level of the user.
- *Point of view* – The comparison method can be general or take the standpoint of a specific framework and view others in terms of that.

How these requirements are satisfied depends on the reason for comparing the SPI frameworks. An organization *without* prior SPI knowledge may wish to institutionalize improvement work because of competitive pressure or certification requirements – but which framework is appropriate? On the other hand, an organization *with* an SPI framework in place may wish to adopt more than one approach – but how can this be done with the least amount of redundancy? In the latter case working knowledge about one specific framework exists, but knowledge about other approaches may not be as thorough.

2. Comparison Methods

There is an increasing amount of literature comparing the major SPI frameworks. Most is written in the last three years and generally cover only a small number of frameworks, e.g. [1][2][3].

From our review of other comparison work we have recognized four main classes of comparison methods. These will be described shortly in the following subsections.

2.1 Characteristics Comparison Method

A comparison method well suited for a general overview is the use of *characteristics.* The characteristics can be nominal, ordinal or absolute and should preferably be objective, measurable and comparable. However, the main point is that they represent areas of interest for the SPI framework investigation.

The frameworks are compared in terms of the defined characteristics and the results can be presented in a tabular format. This gives us a compact and high-level comparison method with little details. Such details must be collected elsewhere, e.g. using another comparison method.

The taxonomy we propose in section 3 is based on the characteristics comparison method.

2.2 Framework Mapping Comparison Method

Framework mapping is the process of creating a map from statements or concepts of one framework to those of another. This requires that the actual frameworks are rather formalized, i.e. consist of a more or less defined set of statements or requirements.

In the characteristics method the goal was to describe important attributes of each SPI framework, i.e. areas of interest. However, the purpose of mapping is to identify overlaps and correlation between frameworks and create a map of these. There can exist strong, weak or no correlation as suggested by Tingey [3]. Furthermore, the mapping can be done on either a high or a low level depending on the amount of detail included. In either case, it is more low-level than characteristics and thus not very useful for a general overview.

Framework mapping is especially useful when an organization employs two or more different SPI frameworks, as corresponding statements can be identified and redundancy reduced. Thus the extra effort needed to employ more than one framework is minimized.

2.3 Bilateral Comparison Method

In a *bilateral comparison* two frameworks are compared textually. The difference between this comparison method and the two previous ones is its textual nature. A bilateral comparison is often a summary or explanation of findings from other the comparison methods.

The bilateral comparison can take on the point of view of one framework and describe another in terms of it. This is convenient for people with detailed knowledge of one framework, because they can easily get insight into another using familiar terms.

The amount of detail included in a bilateral comparison can vary widely, depending on the purpose for which it is written. Frequently the level of detail is somewhere in between that of the characteristics and the mapping approaches.

2.4 Needs Mapping Comparison Method

Needs mapping is not a direct comparison between frameworks. Instead, it considers organizational and environmental needs that must be considered when selecting which SPI framework to adopt. The requirements imposed by such needs are often highly demanding and can limit the choice of framework severely. Nonetheless, they are of utmost importance and must be considered carefully. Here are some examples:

- Certification requirements, for example to ISO 9001, often imposed on a subcontractor.
- Top-level management requires that the chosen SPI approach should be incorporated in a Total Quality Management (TQM) strategy.
- Financial limitations.

There certainly exist other examples as well, and they can vary substantially from organization to organization, or depend on the business environment. Furthermore, the needs may vary over time as the organization or environment evolves.

3. The Proposed Taxonomy

We present a list of 25 characteristics, i.e. areas of interest, relevant for discussing differences between SPI frameworks. Because there are so many characteristics, they have been grouped in 5 categories to enhance comprehensibility and readability (cf. Figure 1).

3.1 General Category

This category describes general attributes or features of SPI frameworks, frequently related to how they are constructed or designed:

- *Geographic origin/spread* – Where did the framework originate and where is it used today?
- *Scientific origin* – The scientific background on which the framework is based, e.g. another SPI framework.
- *Development/stability* – It is desirable to employ an evolved and relatively stable framework. This is achieved through experience feedback from real use over a number of years.
- *Popularity* – A popular framework tends to receive better support and further development than an unpopular framework.
- *Software specific* – Some frameworks are especially geared towards software engineering, others are more general and must be adapted.
- *Prescriptive/descriptive* – Prescriptive frameworks prescribe mandatory requirements/processes. Descriptive frameworks describe a state or certain expectations to be met without assigning specific actions to be taken.
- *Adaptability* – The degree of flexibility in the framework, e.g. does it support tailoring and customization for specific uses?

3.2 Process Category

The process category concerns characteristics that describe how the SPI framework is used:

- *Assessment* – Is an assessment scheme part of the

General	Process	Organization	Quality	Result
Geographic origin/spread Scientific origin Development/stability Popularity Software specifc Prescriptive/descriptive Adaptability	Assessment Assessor Proc. improvement method Improvement initiation Improvement focus Analysis techniques	Actors/roles/stakeholders Organization size Coherence	Quality perspective Progression Causal relation Comparative	Goal Process artifacts Certification Cost of implementation Validation

Figure 1 - Categorization of Characteristics in the Proposed Taxonomy

framework and if so, what is assessed?

- *Assessor* – The assessment can be carried out internally by the organization itself or by an external group.
- *Process improvement method* – What kind of guidelines are included to help implementation and institutionalization of process improvement?
- *Improvement initiation* – Where in the organization is the improvement work initiated, e.g. top-down or bottom-up?
- *Improvement focus* – The SPI activities regarded as the most important by the framework.
- *Analysis techniques* – Does the framework utilize any quantitative or qualitative analysis techniques, e.g. statistical process control or questionnaires?

3.3 Organization Category

The characteristics in this category are directly related to attributes of the organization and environment in which the SPI framework is used:

- *Actors/roles/stakeholders* – Who are the primary people, groups and organizations affected by the improvement process and what roles do they hold in this process?
- *Organization size* – The framework may be more or less suitable for an organization of a certain size, e.g. depending on the required and available resources.
- *Coherence* – Is there a logical connection between engineering factors and factors related to the business or organization[1]? Coherence can exist internally in the organization or externally between the organization and its environment.

3.4 Quality Category

Characteristics in this category are related to the quality dimension of the frameworks:

- *Quality perspective* – The concept of good quality depends on whom you ask, e.g. management, customers or employees.
- *Progression* – Does the framework measure quality progression in a flat, staged or continuous manner?
- *Causal relation* – How does the framework measure an improvement in quality, i.e. what factors are assumed to influence quality?
- *Comparative* – Can the framework be used to compare different organizational units, either internally or externally? If so, which aspects are compared?

3.5 Result Category

The term *result* is loosely used in this category, meaning the outcome originating from the SPI framework adoption:

- *Goal* – The primary objective or end result of using the framework.
- *Process artifacts* – The artifacts created in addition to the actual product as a result of adopting the framework.
- *Certification* – Does the framework include an assessment leading to certification according to ISO or a national standard body?
- *Cost of implementation* – Are there any estimates on how much an adoption and implementation of the framework will cost?
- *Validation* – What kind of validation efforts have been made to evaluate what improvements the framework leads to? Such validation should exclude external success factors, as they would have been achieved even if the SPI framework was not adopted.

4. Conclusion

The goal of comparing SPI frameworks is to provide practical insight and guidance when selecting which SPI framework to adopt in a software-producing organization. Such guidance is needed because of the multitude, diversity and comprehensiveness of existing frameworks. A natural question is whether those SPI efforts that report only a limited degree of success, have adopted the wrong frameworks.

When learning about SPI frameworks it may be necessary to use a combination of comparison methods, preferably starting on a high level. The most interesting frameworks can then be chosen for further investigation, eliminating the costly task to examine all of them.

We believe that our proposed taxonomy is a suitable starting point for such investigations because it describes the most important areas of interest. A major strength of the taxonomy is its compactness, yet it retains the descriptive power of more elaborate comparison methods. However to comprehend the taxonomy fully, some general SPI knowledge is required. There should be no problem collecting material for further investigation, since literature on the various frameworks is vast.

References

[1] Cattaneo F, Fuggetta A. and Sciuto D. *Pursuing Coherence in Software Process Assessment and Improvement.* Paper submitted to IEEE TSE, September 1998.

[2] Sørumgård Sivert, Verification of Process Conformance in Empirical Studies of Software Development. (Doctoral thesis 1997:14, The Norwegian University of Science and Technology, 1997). 252 p.

[3] Tingey Michael O., *Comparing ISO 9000, Malcolm Baldridge, and the SEI CMM for software: a reference and selection guide.* Upper Saddle River: Prentice-Hall, Inc., 1997.

Category	Characteristic	TQM	CMM v1.1	ISO 9000	ISO/IEC 15504	EF/QIP/GQM	SPIQ
General	**Geo. origin/spread**	Japan/World	U.S./World	Europe/World	World/World	U.S./World	Norway/Norway
	Scientific origin	Quality control	TQM, SPC	_[2]	CMM, Bootstrap, Trillium, SPQA.	Partly TQM	TQM, GQM, EF, QIP, ESSI
	Develop./stability	Entire post-war era	Since 1986	Since 1987	Under development	Since 1976	Under development
	Popularity	High (esp. in Japan)	Top (esp. in U.S.)	High (esp. in Europe)	Growing	Medium	Norway only
	Software specific	No	Yes	No	Yes	Yes	Yes
	Prescriptive/ descriptive	Descriptive	Both	Both	Both	Descriptive	Descriptive
	Adaptability	Yes	Limited	Limited	Yes	Yes	Yes
Process	**Assessment**	None	Org. maturity	Process	Process maturity	None	Customer satisfaction
	Assessor	NA[1]	Internal and external	External	Internal and external	NA[1]	Limited internal
	Process improve-ment method	PDCA	IDEAL	None	SPICE Doc. part 7	QIP	Two-level PDCA
	Improvement initiation	Top-down	Top-down	NA[1]	Process instance	Iterative bottom-up	Top-down and iterative, bottom-up
	Improvement Focus	Management processes	Management processes	Management processes	Management processes	Experience reuse	Experience reuse
	Analysis techniques	7QC, 7MP, SPC, QFD	Assessment questionnaires	ISO guidelines and checklists	Several (manual and automated). Required.	GQM	GQM, QFD, 7QC, 7MP
Organ-ization	**Actors/roles/stake-holders**	Customer, employees, management	Management	Customer, supplier	Management	Experience factory, project organization	Customer, experience factory, project org., sponsoring org.
	Organization size	Large	Large	Large	All	All	All
	Coherence	Internal and external	Internal	Internal and limited external	Internal	Internal	Internal and external
Quality	**Quality perspective**	Customer	Management	Customer	Management	All	Customer, all
	Progression	Continuous	Staged	Flat	Continuous (staged at process instance level)	Continuous	Continuous
	Causal relation	NA[1]	**F'**(*Key process areas*) ⇒ **F**(*Maturity level*) ⇒ **Q**(*Process*) ⇒ **Q**(*Product*)	**F'**(*Quality elements*) ⇒ **F**(*Certification*) ⇒ **Q**(*Process*) ⇒ **Q**(*Product*)	**F'**(*Process attributes*) ⇒ **F**(*Capability level*) ⇒ **Q**(*Process*) ⇒ **Q**(*Product*)	**F**(*Experience reuse*) ⇒ **Q**(*Process*) ⇒ **Q**(*Product*)	**F**(*Experience reuse*) ⇒ **Q**(*Process*) ⇒ **Q**(*Product*)
	Comparative	No	Yes, maturity level	Yes, certification	Yes, maturity profile	No	No
Result	**Goal**	Customer satisfaction	Process improvement, supplier capability determination	Establish core management processes	Process assessment	Organization specific	Increased competitiveness
	Process artifacts	Plans, diagrams	Process documentation, assessment result	Process documentation, certificate	Process profile, assessment record	Experience packages, GQM models	Experience packages, GQM models
	Certification	No	No	Yes	No	No	No
	Implementation cost	_[2]	_[2]	_[2]	_[2]	_[2]	_[2]
	Validation	None	Surveys and case studies	Survey	Document review, trials (case studies and surveys)	Experimental and case studies	Experimental and case studies

Table 1 - The Taxonomy Applied to Six SPI Frameworks

[1] Not applicable
[2] Yet to be determined

A Taxonomy of SPI Frameworks

Efficient and Objective Comparison of SPI Frameworks

Christian P. Halvorsen Reidar Conradi

24th NASA SEL Software Engineering Workshop

Software Process Improvement (SPI)

- Product quality depends on the quality of the process used to produce it:

  Quality(Process) $\Rightarrow$ **Quality**(Product)

- Structured in various SPI frameworks, e.g. CMM, ISO 9000, SPICE
- Problem areas between technology and organization

Why Compare SPI Frameworks?

- Practical insight and guidance needed for SPI framework selection
  - Which SPI framework is appropriate?
- Organizational context:
  - No prior SPI strategy in place
  - Implementation of several frameworks

Comparison Difficulties

- Costly and time-consuming
- Knowledge-level of user
  - Appropriate level of detail
- Point of view
  - General or specific?

Classes of Comparison Methods

- Characteristics
- Framework mapping
- Bilateral comparison
- Needs mapping

Characteristics

- High-level/general overview
  - Starting-point for further investigations
- Characteristics should be objective, measurable and comparable
- Purpose: Point out areas of interest when investigating SPI frameworks

Framework Mapping

- Map from statements/ concepts of one framework to those of another
  - Beneficial when several SPI frameworks are used
- Purpose: Identify overlap/correlation between frameworks

Bilateral Comparison

- Textual description
- Can describe one framework in terms of another
- Purpose: Summarize or explain findings from other comparison methods

Needs Mapping

- Identification of requirements from organization or environment
  - May limit choice of SPI framework
- Purpose: Examine external requirements that influence SPI framework selection

Attributes of the Proposed Taxonomy

- Characteristics comparison method
  - 25 characteristics in 5 categories
- Points out areas of interest
- Compact, but retains descriptive power
- Starting point for further investigation
- Applied to TQM, CMM, ISO 9000, SPICE, EF/QIP/GQM, SPIQ

General Category

- Geographic origin/spread
- Scientific origin
- Development/stability
- Popularity
- Software specific
- Prescriptive/descriptive
- Adaptability

Process Category

- Assessment
- Assessor
- Process improvement method
- Improvement initiation
- Improvement focus

Organization Category

- Actors/roles/ stakeholders
- Organization size
- Coherence

Quality Category

- Quality perspective
- Progression
- Causal relation
- Comparative

Result Category

- Goal
- Process artifacts
- Certification
- Cost of implementation
- Validation

Causal Relations in SPI Frameworks

- Process quality difficult to determine
  - Quality indicators
- Causal relations not stated explicitly
- Comparison method influences quality:

SPI Framework	Causal Relation
TQM	Not applicable
CMM	**F'**(*Key process areas*) ⇒ **F**(*Maturity level*) ⇒ **Quality**(*Process*) ⇒ **Quality**(*Product*)
ISO 9001	**F'**(*Quality elements*) ⇒ **F**(*Certification*) ⇒ **Quality**(*Process*) ⇒ **Quality**(*Product*)
ISO/IEC 15504 (SPICE)	**F'**(*Process attributes*) ⇒ **F**(*Capability level*) ⇒ **Quality**(*Process*) ⇒ **Quality**(*Product*)
QIP/GQM/EF	**F**(*Experience reuse*) ⇒ **Quality**(*Process*) ⇒ **Quality**(*Product*)
SPIQ	**F**(*Experience reuse*) ⇒ **Quality**(*Process*) ⇒ **Quality**(*Product*)

F'''(*Comparison method*) ⇒ **F''**(*SPI framework*) ⇒
F'(*Quality indicator*) ⇒ **Quality**(*Process*) ⇒ **Quality**(*Product*)

Category	Characteristic	TQM	CMM v1.1	ISO 9000	ISO/IEC 15504	EF/QIP/GQM	SPIQ
General	Geo. origin/spread	Japan/World	U.S./World	Europe/World	World/World	U.S./World	Norway/Norway
	Scientific origin	Quality control	TQM, SPC	-[2]	CMM, Bootstrap, Trillium, SPQA.	Partly TQM	TQM, GQM, EF, QIP, ESSI
	Develop./stability	Entire post-war era	Since 1986	Since 1987	Under development	Since 1976	Under development
	Popularity	High (esp. in Japan)	Top (esp. in U.S.)	High (esp. in Europe)	Growing	Medium	Norway only
	Software specific	No	Yes	No	Yes	Yes	Yes
	Prescriptive/ descriptive	Descriptive	Both	Both	Both	Descriptive	Descriptive
	Adaptability	Yes	Limited	Limited	Yes	Yes	Yes
Process	Assessment	None	Org. maturity	Process	Process maturity	None	Customer satisfaction
	Assessor	NA[1]	Internal and external	External	Internal and external	NA[1]	Limited internal
	Process improve-ment method	PDCA	IDEAL	None	SPICE Doc. part 7	QIP	Two-level PDCA
	Improvement initiation	Top-down	Top-down	NA[1]	Process instance	Iterative bottom-up	Top-down and iterative, bottom-up
	Improvement Focus	Management processes	Management processes	Management processes	Management processes	Experience reuse	Experience reuse
	Analysis techniques	7QC, 7MP, SPC, QFD	Assessment questionnaires	ISO guidelines and checklists	Several (manual and automated). Required.	GQM	GQM, QFD, 7QC, 7MP
Organ-ization	Actors/roles/stake-holders	Customer, employees, management	Management	Customer, supplier	Management	Experience factory, project organization	Customer, experience factory, project org., sponsoring org.
	Organization size	Large	Large	Large	All	All	All
	Coherence	Internal and external	Internal	Internal and limited external	Internal	Internal	Internal and external
Quality	Quality perspective	Customer	Management	Customer	Management	All	Customer, all
	Progression	Continuous	Staged	Flat	Continuous (staged at process instance level)	Continuous	Continuous
	Causal relation	NA[1]	**F'**(*Key process areas*) ⇒ **F**(*Maturity level*) ⇒ **Q**(*Process*) ⇒ **Q**(*Product*)	**F'**(*Quality elements*) ⇒ **F**(*Certification*) ⇒ **Q**(*Process*) ⇒ **Q**(*Product*)	**F'**(*Process attributes*) ⇒ **F**(*Capability level*) ⇒ **Q**(*Process*) ⇒ **Q**(*Product*)	**F**(*Experience reuse*) ⇒ **Q**(*Process*) ⇒ **Q**(*Product*)	**F**(*Experience reuse*) ⇒ **Q**(*Process*) ⇒ **Q**(*Product*)
	Comparative	No	Yes, maturity level	Yes, certification	Yes, maturity profile	No	No
Result	Goal	Customer satisfaction	Process improvement, supplier capability determination	Establish core management processes	Process assessment	Organization specific	Increased competitiveness
	Process artifacts	Plans, diagrams	Process documentation, assessment result	Process documentation, certificate	Process profile, assessment record	Experience packages, GQM models	Experience packages, GQM models
	Certification	No	No	Yes	No	No	No
	Implementation cost	-[2]	-[2]	-[2]	-[2]	-[2]	-[2]
	Validation	None	Surveys and case studies	Survey	Document review, trials (case studies and surveys)	Experimental and case studies	Experimental and case studies

Table 1 - The Taxonomy Applied to Six SPI Frameworks

[1] Not applicable
[2] Yet to be determined

Discipline of Market Leaders and other Accelerators to Measurement

Stan Rifkin
Master Systems Inc.
PO Box 8208
McLean, Virginia 22106
703/883-2121 Fax: 703/790-0324
sr@Master-Systems.com

Abstract. We often hear that it is difficult to get software measurement into practice. At least one important reason for this is that traditional software measurement is not aligned with the strategic objectives of the organization. When software measurement is aligned with an organization's market discipline then the implementation is accelerated.

One of the reasons it is difficult to get measurement implemented is that it is unaligned with organizational objectives. Measurement is traditionally used to increase quality, increase programmer productivity, and reduce costs. Oddly enough, these are not the highest priority objectives for a number of organizations, so therefore traditional measurement is difficult to implement in them.

The Discipline of Market Leadership is a survey of how 80 organizations out-achieved their competitors. The authors found that focusing on one of three market areas was the answer: operational excellence, customer intimacy, and product innovativeness. Operationally excellent organizations have a "formula" for their service or product. Their menu of choices is small, limited, and with that menu they deliver excellently. Standard examples are McDonalds and Federal Express.

Customer intimate organizations seek quite a different market niche, namely a total solution. Whatever the customer wants gets added to the menu. The menu is long and custom-made for each engagement. Financial service institutions might call customer intimacy a way of getting a greater share of the customer's wallet, there are few spending alternatives outside of the services offered: bank and savings accounts, certificates of deposit, credit and debit cards, travel arrangements, etc.

Product innovative organizations pride themselves on maximizing the number of turns they get in the market. They introduce many new products, selling innovation and features as opposed to, say, price. Examples are Intel, 3M, Sony, and Bell Labs. They measure their success by the number of new product introductions, the number of patents, and/or the number of Nobel prizes.

The authors of *The Discipline of Market Leaders* are quick to point out that all organizations have to have at least threshold characteristics of all three disciplines, but they have to focus on and excel at only one. One example of lop-sidedness cited was IBM's legendary customer intimacy being out-weighed by its inattention to price (that is, operational excellence), so competitors that were not as strong in customer intimacy could gain in-roads to IBM customers with price.

Measurement of the type we are used to, the type espoused by the Software Engineering Institute and Quantitative Software Management, applies almost exclusively to organizations wishing to be operationally excellent. We typically have nothing to offer to customer intimate and product innovative firms in our measurement or improvement methods.

Many software development organizations do not strive to become operationally excellent, so we have left them in the lurch, though we tend to treat them as resisters and of bad character! In fact, it is nothing more than a mismatch of goals. There is, for example, a large set of software development organizations that strive for customer intimacy and essentially will do anything their

clients request. Those organizations get to know their clients very, very well, sometimes better than the clients knows itself. An example of this might be a payroll service that has seen every variation on payroll and knows more about payroll processing than any in-house payroll department could. The most customer intimate payroll service offeror would take over their customers' payroll departments!

What do you think Microsoft's market discipline is? I think it is product innovativeness. It touts its new, glitzy features, not its up-time or reliability. It wants to own/earn its clients based on new features, not offering software that is operationally excellent. In that context, the Software Engineering Institute's Capability Maturity Model for Software is silent on product innovativeness and customer intimacy: it applies only to organizations wanting to be operationally excellent. Same for traditional measurement.

What are we missing in all of this? A more global view, one that listens to and responds to our measurement customers. We need to see that the potential rejection of our measurement efforts is NOT an indicator of bad character or resistance, but may be an appropriate response to measures that do not fit the strategy. We need to joint problem-solve with our clients to develop new classes of measures that simultaneously meet our high standards for objectiveness and their high standards for relevance.

Examples. Let me relate several efforts in which I have participated:

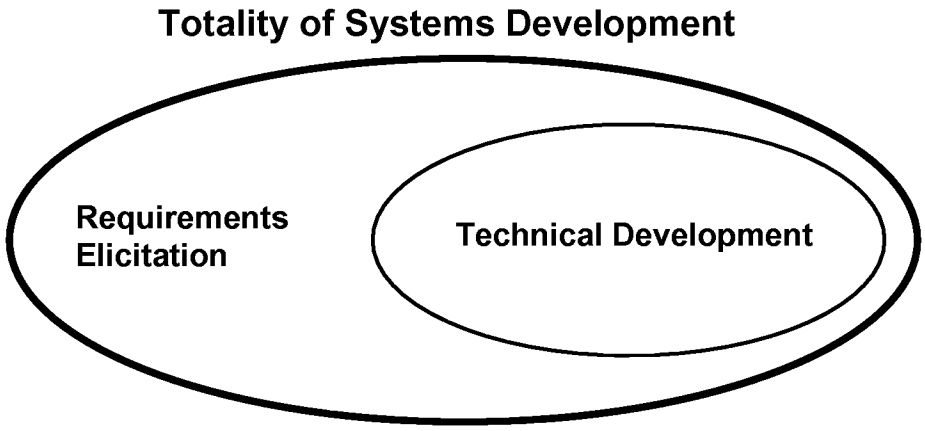
1. One brokerage house was not interested in software costs or quality, but rather what it called time to market. In fact, it was not speed that was so important, but rather during the frantic time that a deal (such as an initial public offering) was being put together Information Technology was being asked to respond quickly. The response had to be quick enough so that the broker could earn as much as possible by offering as many services as possible. It was a question of wallet share, which in turn is a customer intimate approach. The brokerage wanted the customer to maximize its spending with the brokerage so it had to have the longest menu of services possible. We settled on a measure of the percentage of the total deal that did not go to the brokerage. I/T's job, then was to offer a realistic plan for continual reduction of that (missed wallet share) figure.
2. One computer-oriented defense contractor said it wanted project measures, but when pressed it was clear that projects were not managed - and therefore not measured - in the traditional way. The government client wanted a provider that would do what it requested, not study the request and offer alternatives or push-back. Cost, quality, and duration were not important to the client, only that it got what it wanted in reasonable terms. This, too, is a customer-intimate approach, one that makes the menu of services just as long as the customer requests are. Naturally, the provider has to deliver the systems within a threshold value of cost, quality, and duration, but there were already many other providers that performed better in terms of cost, quality, and duration, but were rated too low in customer responsiveness to be considered! In fact, the client changed its mind often, rendering previous work inapplicable. This would cause rework that would traditionally be held against the provider. Traditional project-oriented measurement was irrelevant in this setting. We recommended several measures: of the total spent by the customer how much went elsewhere (to be minimized); time spent in adversarial settings (to be minimized); time spent with the customer understanding its business (to be maximized); and number of people on our staff with credentials like our client's (to be maximized).
3. A computer services firm had been the prime contractor for a long time for a government client. The computer services firm provided all of the computer programming and operations for a particular type of payment that the government entity made to deserving applicants. The contract was up for renewal and the incumbent wanted to propose a set of measures going forward that would indicate its operational excellence. The usual suspects were offered in discussions with the provider (now bidder), but those measures did not seem to resonate, even though they were "reasonable." It turns out that the government organization was feeling behind the times in terms of technology and really wanted a new, modern I/T provider, not a better, cheaper, faster provider of old technology. In fact, there was no business driver for the

desire for more modern technology, only a (vague) belief that such technology would reap financial benefits to the government in terms of lower costs and greater flexibility. The measures we settled on were:

- plan vs. actual implementation of a set of new technology introductions,
- hours spent training the government client on the principles of that new technology,
- reliability measures directly related to the government organization's business, for example, cost of government rework due to provider payment errors, idle government worker hours due to system downtime, and government time spent in meetings or on the phone with deserving applicants due to provider service failures.

These measures were instead of other, traditional measures, such as percentage system availability (e.g., 99.9% available), data entry error rates (0.1%), and a threshold number of ABENDs per day, none of which related to the government mission or daily reality.

New model emerging. There is a new model of systems development emerging. It is consistent with the lessons of *The Discipline of Market Leaders*. The place it is first seen is a new breed of systems developers: fixed price, fixed duration efforts. Their model is something like:



The totality of the systems development effort is internally divided into two phases: obtaining customer requirements and developing a system to meet those requirements. Obtaining requirements is an open-ended effort, difficult to estimate, and bid on a time and materials basis. Once the requirements are obtained, they are more or less thrown over a wall to a heads-down software factory. There the requirements are quickly transformed into an operational system.

Changes to the requirements are not allowed during the factory period. It isn't that changes are not requested, but rather they are queued and made candidates for the next release. A small percentage of high-priority changes can be accepted and passed on to the factory, but usually it is a single digit percentage, by contract agreement.

Because the factory can work with its head down, it is fast and good. It has learned how to be, perhaps by emulating/applying the best practices promulgated by the NASA-CSC-University of Maryland Software Engineering Laboratory, the SEI, Cleanroom, and others.

If the requirements need to change dramatically before the system is developed, then the whole arrangement changes back to what we traditionally have today: gather requirements, try to freeze (or at least chill) them, develop a system to meet the requirements, in the midst of that then change requirements again and try to absorb the newest changes, etc.

The newer* business model achieves several objectives:

1. People attracted to dealing with customers face-to-face do that and only that.
2. People attracted to dealing with the technical development of systems do that and only that.
3. People who like to span those boundaries get to do that, too, because they are part of both the requirements elicitation and the technical systems development so that, in fact, the requirements are not thrown over a wall.

Technology improvement and change management are different in the two areas. Technical development is the stuff we are used to seeing, but the technology of customer intimacy used in requirements elicitation is new, both in the technical aspects and in how change is introduced into a on-going working relationship between technology solution providers and their clients.

I see ever more organizations offering this newer model. What would be the downside to the client? Anyway, the measurement implication is that a wholly different set of measures would apply to the customer intimate activity than to the technical one; the technical one is more or less a solved problem. Now we as a profession need to turn to the other two disciplines of market leaders and offer them something!

Acknowledgements. I learned most of this by working with John Title of Computer Sciences Corporation. The measurement leader who made me ask myself many of these questions is David Card. Many SEI SEPG conference keynote speakers/cheerleaders who claim that those who resist have bad character have irritated me into writing this down. Their failure to ask (and answer) "Why?" stimulated me.

References.

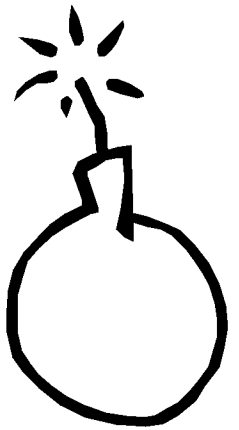
Michael Treacy and Fred Wiersma. (1995). *The Discipline of Market Leaders: Choose Your Customers, Narrow Your Focus, Dominate Your Market*. Addison-Wesley.

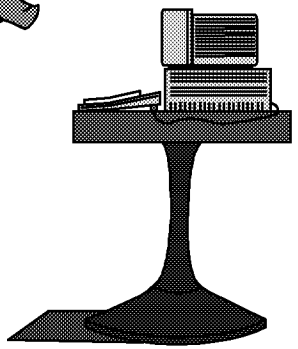
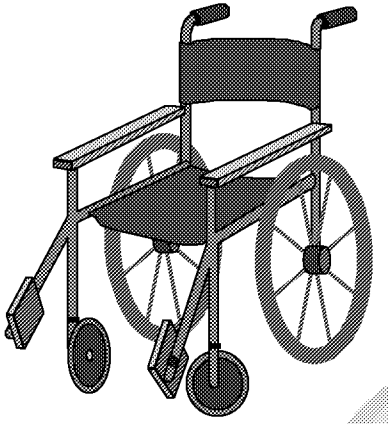
Fred Wiersema. (1996). *Customer Intimacy: Pick Your Partners, Shape Your Culture, Win Together*. Knowledge Exchange.

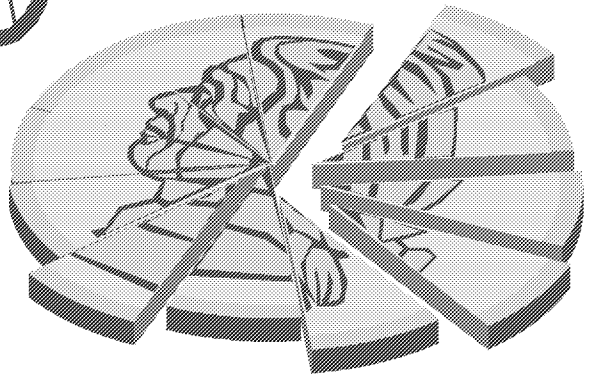
* I hesitate to call it "new" because Winston Royce, that doyen of our profession, described it in a keynote address at the National Security Industrial Association Seventh Annual National Joint Conference, April 23, 1991, at Tyson's Corner. His talk was entitled, "A completely new software life cycle."

Discipline of Market Leaders and Other Accelerators to Measurement

Stan Rifkin
Master Systems Inc.
PO Box 8208
McLean, Virginia 22106
USA
+1 703 883 2121 +1 703 790 0324 fax
sr@Master-Systems.com

Discipline of Market Leaders

- by Treacy & Wiersema
- Survey of 80 high performing firms
- Key to success: Focus

Operationally Excellent

- **Highest quality => lowest cost**
- **"Formula" => short menu**
- **Process innovative**

FedEx

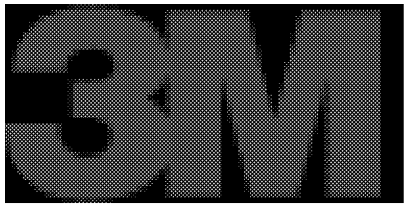
MOTOROLA

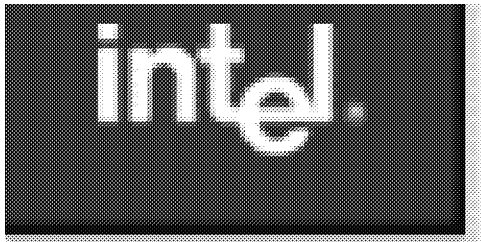
WAL★MART

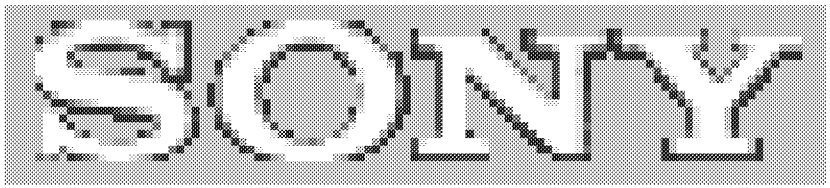
TEXAS INSTRUMENTS

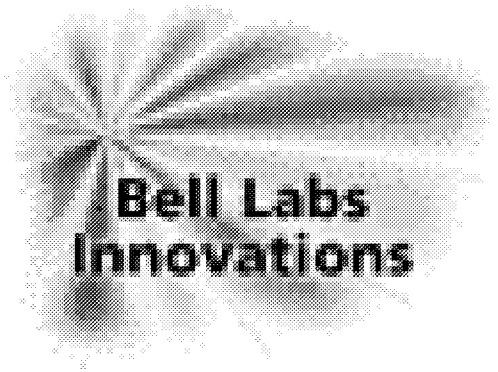
Product Innovative

- **Market leader in innovation**
- **Measure: number of patents, Nobelists, turns in the marketplace**



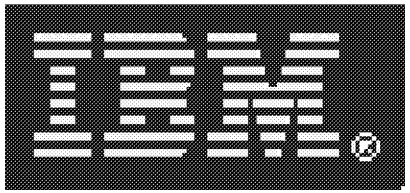


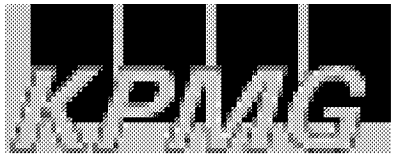


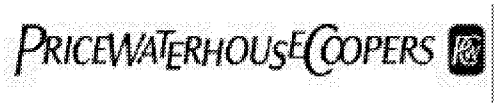


Customer Intimate

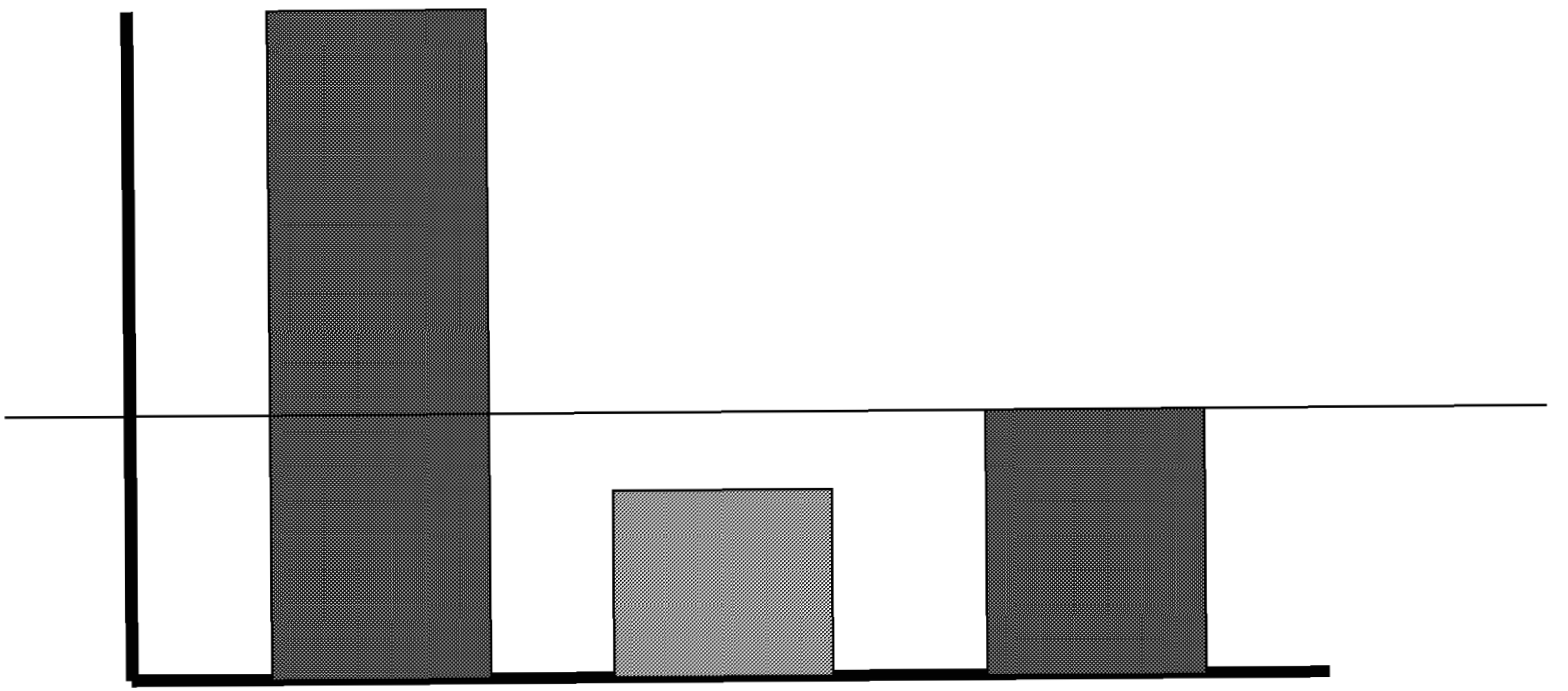
- "Infinite" menu
- Measure: "walletshare"
- NOT lowest cost, highest quality, most innovative
- "Schmoozes"







Focus on one, have to meet the threshold in all



Square peg - round hole

Operationally Excellent

- **Highest quality => lowest cost**
- **"Formula" => short menu**
- **Process innovative**

FedEx

MOTOROLA

WAL★MART

TEXAS INSTRUMENTS

Customer intimate: flexibility is key

- What is measured: Wallet-share
- How do you get wallet-share: by offering the most options to help the customer say “Yes”
- Look for chances to be flexible, that do not foreclose options

Product innovative: features are key

- **CMM KPA Goal 1: "*xx* is planned"**
- **Plan: "1.4 breakthroughs per fortnight"**
- **Instead - risk manage and create an environment of creativity (= OK to fail *in the small*)**
- **Lightweight processes**
- **"Good enough quality"; *appropriate* benchmarking is imperative**

Align measurement with strategy!

- **“Fit” is an important, practical reality**
- **Always ask “Why?”**
- **Remember: we are in one of the slowest moving and least understood professions, so**
- **Stay alert, don’t believe everything you hear/read**
- **Search, seek, keep high standards!**

Software Measurement Frameworks to Assess the Value of Independent Verification & Validation

Dr. Nancy Eickelmann
NASA IV&V Facility
Software Research Laboratory
100 University Drive
Fairmont, West Virginia 26554
+1 304 367 8444
http://research.ivv.nasa.gov/~ike
Nancy.Eickelmann@ivv.nasa.gov

Abstract

Software IV&V, as practiced by the NASA IV&V Facility, is a well-defined, proven, systems engineering discipline designed to reduce risk in major software systems development. However, we currently have no proven methodology for estimating resource requirements for IV&V based on sound financial criteria. The quantification of a cost structure associated with IV&V and the resulting benefits are essential to make objective decisions concerning the allocation of resources to IV&V activities. The development of ROI metrics for NASA IV&V would provide key information to make rational budgetary decisions that impact safety and mission critical aspects of all NASA software systems. To measure IV&V benefits and costs we must identify relevant measures and provide target ranges for those measures that may be used to evaluate whether or not the goals are achieved and to what degree. This requires a measurement strategy for software IV&V in the NASA context. This paper presents the NASA IV&V Balanced Scorecard strategic measurement framework and discusses its role in providing a minimal and usable core metrics set.

1 Introduction

The Balanced Scorecard, as applied in industry and government, is approached from two very disparate viewpoints. Industry is very aware of the importance of financial performance measures in managing an organization. Publicly held companies must be responsive to market and shareholder demands. Market share, share price, dividend growth, and other significant results-oriented financial measures have been used historically to evaluate an organization. Government organizations must respond to regulatory and legislative acts. One such legislative act is the Government Performance and Results Act (GPRA) passed by Congress and signed by the President in 1993. This act provides a new tool to improve the efficiency of all Federal agencies.

The goals of GPRA are to:

- Improve Federal program management, effectiveness, and public accountability
- Improve congressional decision making on where to commit the Nation's financial and human resources
- Improve citizen confidence in government performance

A specific difference between government and industry is explicit in the government's focus on cost reduction as compared to industry's focus on revenue generation and profitability. We have customized our BSC to accommodate these differences thus providing a framework to evaluate the overall performance of the organization through a linked hierarchy of specific performance drivers and outcome measures [7].

1.1 Structure of the Paper

Section 2 provides an overview of the Balanced Scorecard and motivations for its use. We then excerpt portions of our scorecard to exemplify our measurement framework, the application of cause effect graphing and the setting of strategic measurement targets in Section 3. Section 4 discusses specific BSC measurement issues and lesson learned. Section 5 concludes our paper and discusses current directions of our work.

2 Balanced Scorecard

The Balanced Scorecard (BSC) Framework provides the necessary structure to evaluate quantitative and qualitative information with respect to the organization's strategic vision and goals. There are two categories of measures used in the BSC the leading indicators or performance drivers and the lagging indicators or outcome measures. The performance drivers or leading indicators enable the organization to quantitatively track whether or not the organization is achieving short-term operational improvements. The outcome measures or lagging indicators provide objective evidence of whether *strategic objectives* are achieved and to what degree. The two measures must be used in conjunction with one another to link measurement throughout the organization thus giving visibility into the organizations progress in achieving strategic goals through process improvement [14].

The development of a core set of metrics for implementing the Balanced Scorecard is the most difficult aspect of the approach. Developing metrics that create the necessary linkages of the operational directives with the strategic mission prove to be fundamentally difficult as it is typical to view organizational performance in terms of outcomes or results rather than focus on metrics that address performance drivers that provide feedback concerning day-to-day organizational progress.

The BSC is not the organizational strategy but rather a measurement paradigm to provide operational and tactical feedback. The organizational strategic vision and goals are the foundation upon which the framework is constructed and are taken from public domain documents. The strategic plan contains the vision, goals, mission and values for the organization. The Government Performance and Results

Act, GPRA requires all federal agencies to establish strategic plans and measure their performance in achieving their missions. The vision and goals are stated below.

> Vision: To be world-class creators and facilitators of innovative, intelligent, high performance, reliable informational technologies that enable NASA missions.
>
> Goals: To become an international leading force in the field of software engineering for improving safety, reliability, quality, cost and performance of software systems; and to become a national Center of Excellence (COE) in systems and software independent verification and validation.

3 BSC Architecture

The BSC architecture was intended to provide a framework for industry and for-profit organizations. The framework facilitates translating the strategic plan into concrete operational terms that can be communicated throughout the organization and measured to evaluate its day-to-day viability. The three principles of building a balanced scorecard that is linked through a measurement framework to the organizational strategy include;

(1) defining the cause and effect relationships,
(2) defining the outcome measures and performance drivers,
(3) linking the scorecard to the financial outcome measures [5].

The initial steps of BSC engage in the construction of a set of hypotheses concerning cause and effect relationships among objectives for all four perspectives of the balanced scorecard. The measurement system makes these relationships explicit. Therefore, they can be used to assess and evaluate the validity of the BSC hypotheses. The questions asked in each category of the four perspectives provide a segue into the cause effect diagramming activity. It is this activity that exposes the value chain associated with specific IV&V activities.

3.1 Defining the Cause-Effect Relationships

IV&V is conducted using different approaches and methods depending the goals of the IV&V team. To define causal relationships we must evaluate the measurement based on a context sensitive method:

1) Identify the underlying IV&V process relative to the development process.
2) Identify the activities (methods, models and tools) by inputs and outputs and entry and exit criteria.
3) For activities categorized as information management IT, measure the value of information to decrease uncertainty, mitigate risk, improve quality...
4) For analysis activities we define the value for the outputs such as problem reports at a given time in the lifecycle and by criticality.

We begin by formulating hypotheses concerning the value of IV&V in a given context of the Space Shuttle IV&V activities. The hypotheses are based on inferred or known relationships documented in prior studies reviewed under the first phase of our ROI project. We state the initial hypotheses as constructed, however their review and evaluation are an ongoing activity.

The hypotheses developed are based on several assumptions that are based on current understanding of the interaction of the IV&V process and shuttle development process. The Space Shuttle is considered a product-line as defined by the SEI as well as the general research community. The characteristics that make the shuttle a product line process include the systematic reuse of a set of core architectural and component based assets that are reused in each incremental release. This core commonality is extended to support each operational increment (OI) and represents a negotiated and limited degree of domain variability.

> Hypothesis 1: The benefits of IV&V contributions are realized as domain engineering and applications engineering benefits. This means some benefits should accrue to the core structure of shuttle software and be an ongoing contribution in its maintenance and extensibility.
>
> Hypothesis 2: The benefits of the application engineering accrue almost entirely to the developer. That is the defect reduction that occurs in development is enabled in part by IV&V contributions to domain engineering.
>
> Hypothesis 3: The benefits of product-line engineering in the shuttle are significant in reducing testing costs while maintaining high levels of testing quality. The degree of test suite and test environment reuse is exceptionally high and results in a significant cost savings.
>
> Hypothesis 4: This is fundamentally a unique system that is developed using sophisticated reuse. This requires us to view the system as generating shuttle "builds" from an investment of core assets. The benefits are primarily derived in the reusability and rapid extensibility of the shuttle code.
>
> Hypothesis 5: Adherence to an architecture enables system safety, reliability and quality standards to be imposed and verified for the core assets of the shuttle. Acceptable degrees of variability to extend functionality are approved by a team of architects and systems engineers that includes the IV&V team.

We map our hypothesis to a set of objectives concerning the value of IV&V and the necessary and sufficient factors to creating value for the organization in terms of the strategic vision and goals. The BSC is segmented into four categories of objectives customer, financial, internal business processes and learning and growth segments. The objectives for the four segments are the following:

- customer segment objectives correspond with the high level goals of mission success through high quality, reliability and safety.
- financial segment objectives focus on cost reduction, efficient asset utilization and high ROI values of IT investments.
- internal process objectives relate to specific software and systems engineering approaches such as product-line development paradigms, CPI and QIP efforts, and test technologies and best practices as defined for IV&V.
- learning and growth objectives include technological infrastructure for distributed development, workforce training programs, skills assessment program, and ISO-9000 process structure.

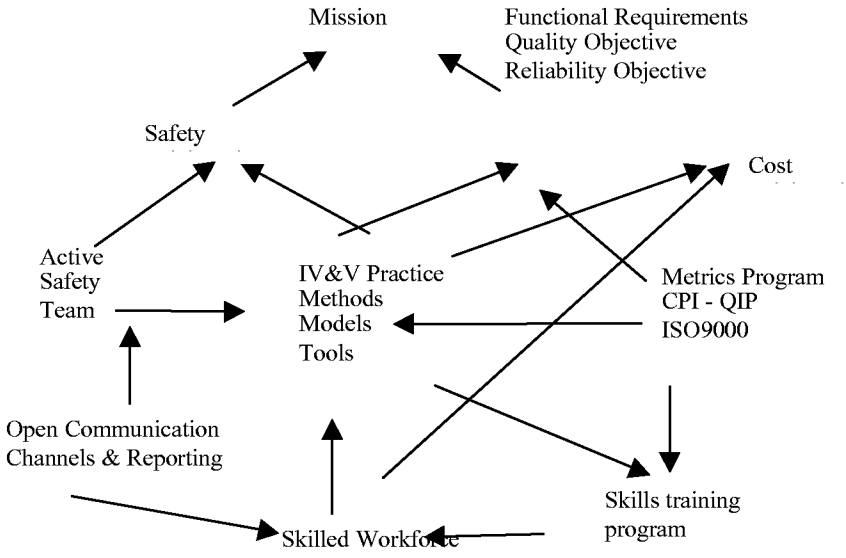


Figure 1.1 Influence diagram of IV&V BSC objectives.

The objectives are used in the selection of a minimum set of required metrics to measure day-to-day performance as well as longer term outcome or results metrics. This aspect of the framework focuses on development of leading and lagging indicators. An example customer focused objective would be the improvement in overall safety due to IV&V activities. A leading indicator for this objective could be the number of identified potential hazardous states resulting from a safety impact analysis or a tracking of the hazard rate during development. A result measure or lagging indicator could be the number of in flight anomalies (IFA) that are documented. The leading and lagging indicators must be assigned desired or normative values. These values become targets or target ranges for the metrics collected. Finally, the initiatives that have been sponsored to achieve the objective is identified and reevaluated with respect to the quantitative and qualitative evidence of success relative to the target values (see table 1.1.)

	Objectives	Measures	Targets	Initiatives
Customers (Internal External)	No Losses	# Severity 1 &2	Remove < FRR	Formal Methods
	Reduce Risk	# IFA's	No Severity 1	Risk Management
	Manage Risk	Fault tolerance	Performance	Risk Mitigation

Table 1.1 Customer focus metrics definition.

The relationships among the customer objectives of interest are significant as they are not independent of one another and therefore must be analyzed based on their degree of covariance and interaction. The relationships are diagrammed Fig.1.3 and depict the current accepted understanding. Safety requires that unsafe states cannot be entered from any point of function of the system. It is possible for the systems to function reliably that is without failure and still enter unsafe states of operation. A system can be completely correct and defect free and still enter unsafe states. There are many documented examples of these properties in the literature and many devoted specifically to documenting the complexity of software safety issues. The safety of a system is a result of its safe operation in a specific context or environment. We provide definitions of safety, reliability, quality and cost as defined for the customer objectives of the BSC.

- Safety is defined as freedom from accidents or losses. This is an absolute statement, safety is more practically viewed as a continuum from no accidents or losses to acceptable levels of risk of loss.
- Reliability is defined in terms of the probabilistic or statistical behavior, that is the probability that the software will operate as expected over a specified period of time.
- Quality is defined in terms of correctness and number of defects. Correctness is an absolute quality, it is also a mathematical property that establishes the equivalence between the software and its specification.
- Cost is more complex than it appears, direct or absorption costing may be applied and alters what costs are included and therefore what costs may be reduced. The focus of the paper does not rely on the differences inherent to these two approaches and therefore defers discussion of this topic.

The NASA IV&V facility must document the increase in software and systems safety, reliability and quality that are attributable to IV&V technologies. This requires that the contribution that is made towards meeting required targets through the application of IV&V activities must be quantified. This requires that each aspect be evaluated relative to some objective target. The value add of IV&V is measured as the sum of overall reduction of distance from the target. This provides a measure of overall impact to mission success. The relative reduction of "Euclidean Distance" from the safety target of no losses attributable to IV&V specifically is documented and integrated into the overall model that sums the total reduction of distance from the three targets of safety, reliability and quality. There are many measures that can be collected to evaluate the value added of IV&V for software and system safety; this is only one approach. The measurement of the contribution of IV&V in improving safety, reliability and quality while reducing cost is discussed in the following sections.

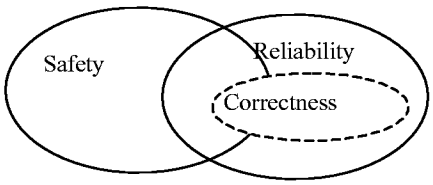


Fig.1.3 Relationships among customer themes of mission success through safety, reliability, and quality at reduced costs.

4 BSC Issues and Lesson Learned

The four strategic mission goals of importance to our customers are safety, reliability, quality and cost. This section discusses those aspects in terms of measurement as is defined in the balanced scorecard.

SAFETY The contribution of IV&V to shuttle safety is difficult to measure directly. It is therefore necessary to make assumptions concerning those factors that would impact safety and to what degree. It is assumed that a reduction in the probability of failure is a contribution to increased safety. A reduction of the number of In Flight Anomalies IFAs of a severe nature due to IV&V identification and removal is a contribution. An independent evaluation of potential failure modes that results in identifying previously unidentified hazards is a contribution.

RELIABILITY The contribution of IV&V to shuttle reliability is more directly attributable to the specific verification activities that are applied during the Shuttle software development process towards defect management. Research investigating the ramifications of testing strategies for reliability provides quantification of benefits relative to specific IV&V activities. A minimization of estimated residual faults is provided according to the sequence of testing strategies and the duration of those test executions. For example the number of defects detected by applying functional, decision, data flow and mutation test methods in sequence. The CPU execution time or the number of test cases can measure test effort. As the test effort increases defects detected can be optimized through applying more optimistic or pessimistic test strategies. The resulting increase in reliability is measured by increased MTTF or improved failure intensity profiles and is quantified as a reduction in the distance from the reliability targets of subsystems undergoing IV&V.

QUALITY The contribution of IV&V to shuttle quality is measured as a reduction of defect density trends through process improvement paradigms such as traversing the CMM stages from levels 2,3,4 to level 5. The intuition behind this model is that the measurable impact of process improvement is in the reduction of the cost of rework Specific examples of applying this concept are documented in the literature and state substantial savings associated with rework avoidance. Raytheon Systems Corporation reported cost savings of $15.8 million for 15 projects over a four-year period. Raytheon documents an ROI of 7:1 based on $4.48 million return for $580,000 invested. Hughes Aircraft reported cost savings of $9.2 million over a three-year period. Hughes documents an ROI of 4.5:1 based on $2 million return on $400,000 invested. The Aircraft Software Division at Tinker Air Force Base reported an ROI of 6.35:1 based on a return of $2.9 million for $462,100 invested. In addition, the rework cost avoidance of detecting defects of severity 1; severity 2 and severity 3 can be quantified relative to phase of detection and level of severity. The reduction of defect density is measured as a reduction of distance from the overall quality objective measured in defect density according to severity.

COST In the early 1990's the software engineering community adapted ROI to measure the costs and benefits of SEI/CMM process improvement efforts. Published examples of how ROI for CMM based process improvements are measured and interpreted provide guidelines for the basic proposed ROI model [7,13]. The process

community quantified process and product improvement using the following four major development-cost structures drawn from Crosby's work as published in "Quality is Free" and "Quality Without Tears" [3,4]. Crosby's work is referenced by Capers Jones as the seminal work in this area and has been used as the basis for cost structuring by DoD contractors such as Raytheon Systems [17]. The cost categories include:

1. nonconformance rework costs (such as fixing code defects or design documentation),
2. performance costs associated with doing it right the first time (such as developing the design or generating the code),
3. appraisal costs associated with testing the product to determine if its faulty, and
4. prevention costs incurred trying to prevent faults from degrading the product.

Industry has applied these four cost categories to measuring ROI for software process improvement by using rework costs avoided (nonconformance costs avoided) as the numerator and appraisal and prevention costs directly related to process improvement efforts for the denominator [7,18]. The intuition behind this model is that the measurable impact of process improvement is in the reduction of the cost of rework [3,4,10,11].

A measurement framework is necessary to bridge the gap between strategic measures of improved reliability, safety, and quality at reduced cost and operational measures of optimization of resource allocations applicable to daily activities to achieve these goals. The BSC provides a means of measuring the efficiency of resource allocations for the operational processes of software and systems verification and validation activities that must then be linked to the high level goals of mission success at reduced cost. In applying the BSC we have learned many lessons of value concerning our strategic planning as it relates to the activities conducted to accomplish daily operational goals. First, we have found that a customer focus of the strategic themes provides the necessary linkages in the BSC to measure our leading and lagging indicators successfully. We have also learned that the CMM and ISO-9000 initiatives are split across the core process tier and the infrastructure tier of the BSC hierarchy. These two findings are essential in applying the BSC to a government or not-for-profit organization such as the NASA IV&V Facility.

5 Future Directions

The primary focus of learning and growth measures for IV&V specifically is the information technologies (IT) used to obtain, retrieve, disseminate and store key information products [6]. The IV&V Facility is located in West Virginia and yet services all the NASA Centers from the Pacific to Atlantic coasts. To support this distributed context. Communications technologies such as VITS, VOTS and internet tools such as web-based data collection repositories are required. Specific measures to quantify performance, cost, and quality for IT infrastructure to support IV&V technologies must be further evaluated to provide meaningful target ranges for IT performance metrics.

In addition, further investigation into the measurement of core processes as defined under ISO is required. The ISO-9126 Standard, documents 6 high-level software qualities including functionality, reliability, usability, efficiency, maintainability and portability. These high-level qualities are mapped to 24 sub-characteristics. Metrics are proposed to measure the high-level software qualities relative to the sub-characteristics. This ISO standard could provide the necessary metrics to measure operational processes under the process aspect of the BSC, relative to the application of product line reuse, and map them to the high-level goals. Of particular interest in this standard is the definition of reusability as the combination of maintainability and portability. It will be of interest to analyze the appropriateness of the standard in measuring reuse for the shuttle [9]. Specifically, reuse across a vertical product line that incorporates domain engineering, architecture-based reuse, and reusable test technologies.

REFERENCES

[1] Basili, V. Rombach, D., "The TAME Project: Towards Improvement Oriented Software Environments," IEEE Trans. Software Engineering, 1988.

[2] Boehm, B., Software Engineering Economics, Englewood Cliffs, Prentice Hall, 1981.

[3] Crosby, P. B., Quality is Free. McGraw Hill, 1979.

[4] Crosby, P. B., Quality without Tears. McGraw Hill, 1985.

[5] Eickelmann, Nancy S., "Combining Software Measurement Frameworks to Assess the Operational and Strategic Value of Process Improvement in a Government Organization"European Software Process Improvement Conference: Learn from the past – experience the future. In the Proceedings of the EuroSPI '99 at the Pori School of Technology and Economics, Pori, Finland, Oct. 25-27, 1999.

[6] Eickelmann, Nancy S., " Strategic and Software Measurement Frameworks to Assess the Value of Information Technology", In the Proceedings of FESMA '99 European Software Measurement Conference, Amsterdam, Netherlands. Oct. 4-8, 1999.

[7] Eickelmann, Nancy S., "A Comparative Analysis of BSC as Applied in Government and Industry Organizations." Information Technology Balanced Scorecard Symposium, Antwerpen, Belgium, March 15-16, 1999.

[8] Eickelmann, Nancy S., "Measuring and Evaluating the Software Test Process." European Software Measurement Conference, FESMA `98, Antwerp, Belgium, May 6-8, 1998.

[9] Eickelmann, Nancy S., Product-Line Development Metrics. GSAW `98, El Segundo, California, February 25,1998.

Hetzel, B., Making Software Measurement Work. John Wiley and Sons, 1993.

[10] Humphrey, W., Managing the Software Process. Addison-Wesley 1989.

[11] Humphrey, W., Snyder, T., and Willis, R., "Software Process Improvement at Hughes Aircraft," IEEE Software, July 1991.

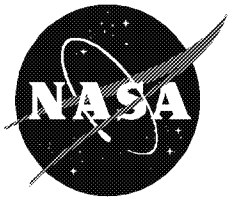
[12] Jenner, M., Software Quality Management and ISO 9000. John Wiley and Sons, 1995.

[13] Jones, C., Applied Software Measurement. McGraw Hill, 1991.

[14] Kaplan, R. and Norton, D., The Balanced Scorecard. Harvard Business School Press, 1996.

[15] McGrath, R. and MacMillan, I., "Discovery-Driven Planning" Harvard Business Review, July-August 1995.

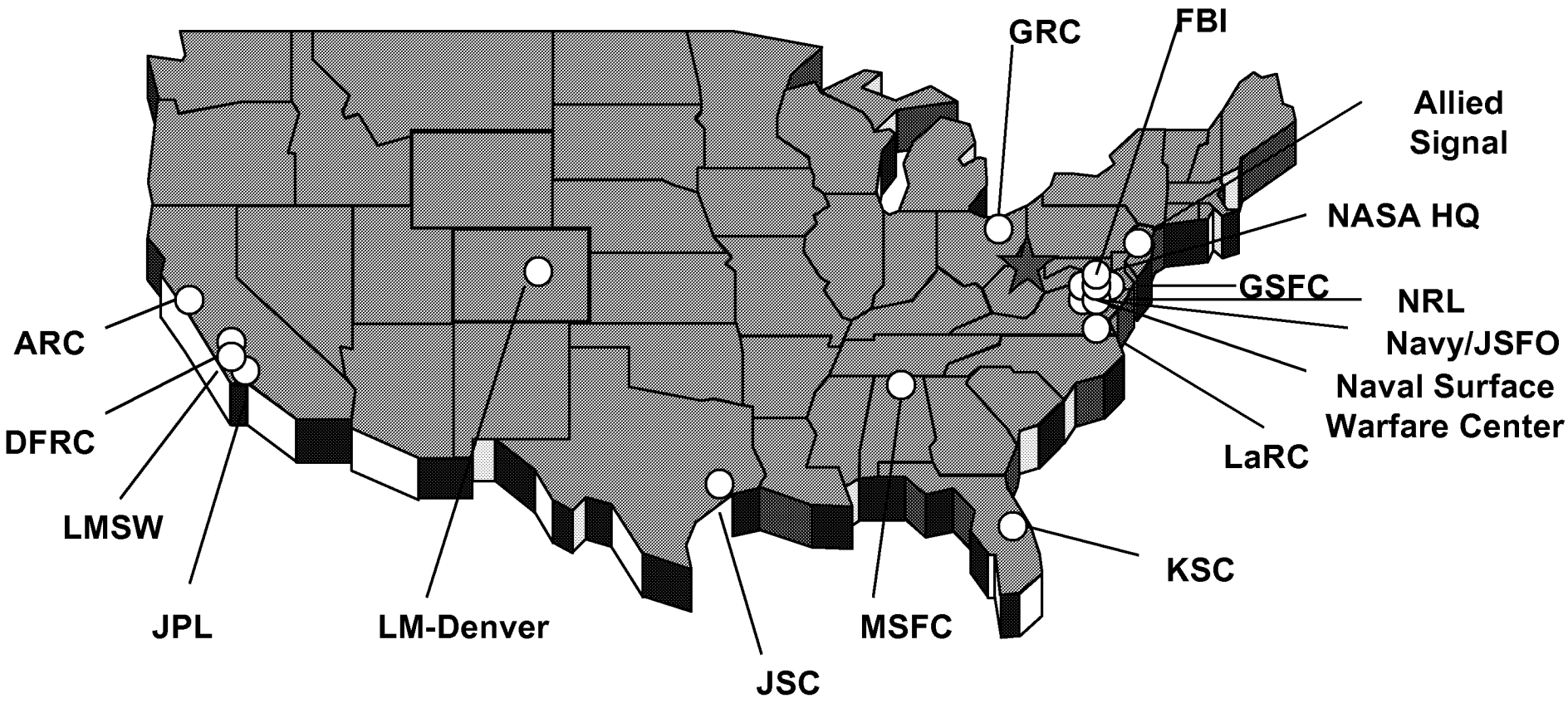
[16] Radatz, J. W., "Analysis of IV&V Data" Rome Air Development Center ROME C# F30602-80-C-0115, 1981.
[17] Saiedian, H. and Kuzara, R., "SEI Capability Maturity Model's Impact on Contractors" IEEE Computer, January 1995.
[18] Violino, B., "Measuring Value: Return on Investment" Information Week, Issue 637, June 30, 1997.



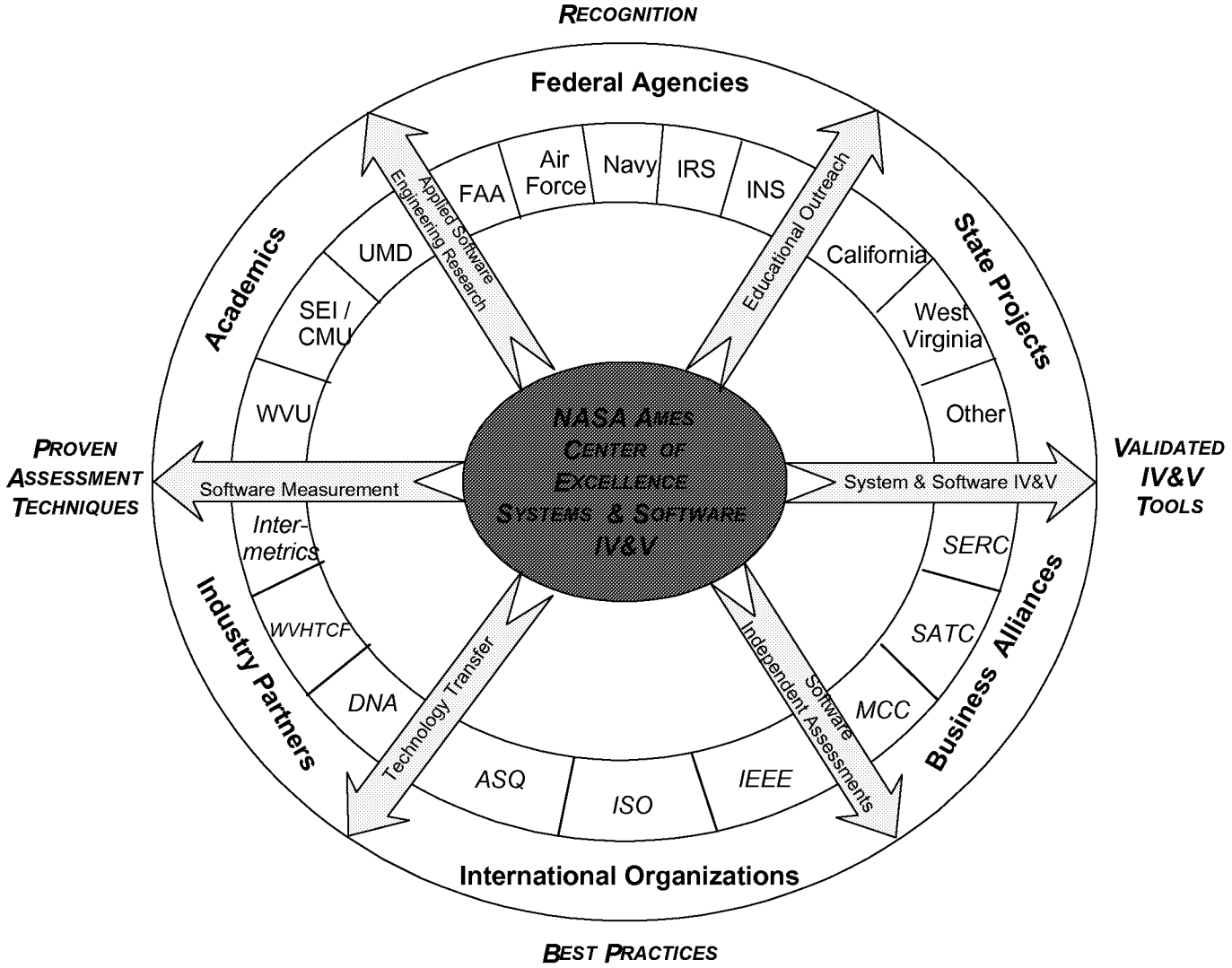
Software Measurement Frameworks to Assess the Value of Independent Verification and Validation

Dr. Nancy S. Eickelmann
WVU/Software Research Laboratory
NASA Independent Verification and Validation Facility
Phone: (304) 367-8444
E-mail: Nancy.Eickelmann@ivv.nasa.gov
Website: http://research.ivv.nasa.gov/~ike
December 1999

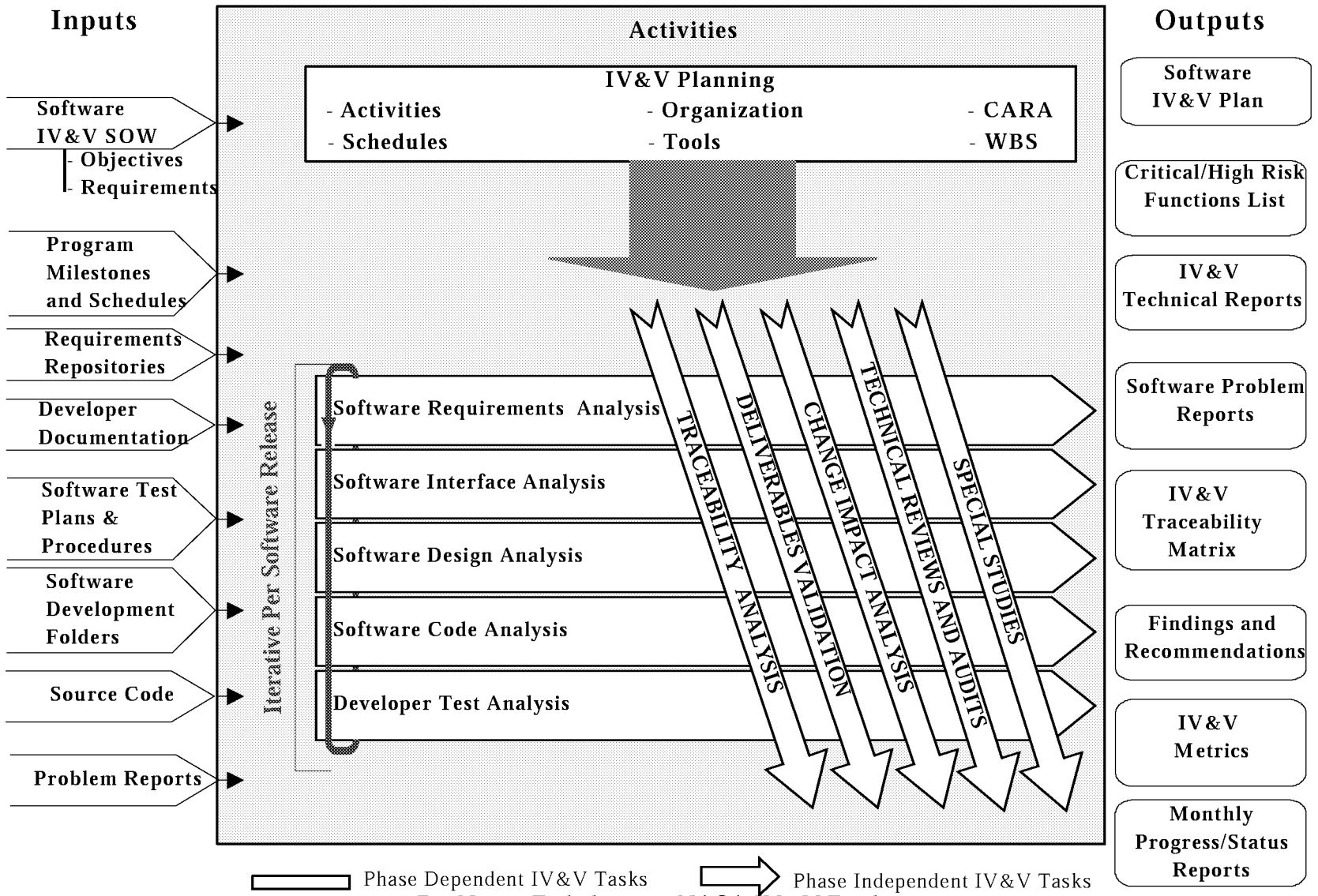
IV&V Infrastructure Supports Programs Across Country



NASA IV&V Core Customers, Processes and Practices

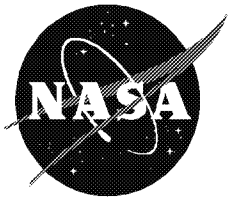


Iterative IV&V Methodology



Dr. Nancy Eickelmann - NASA IV&V Facility

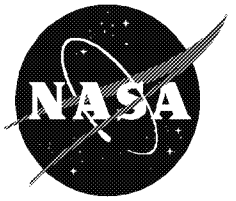
IV&V Goals



- IV&V processes provide an objective assessment of software products and processes throughout the software life cycle.
- This assessment demonstrates whether the system and software requirements are ***correct, complete, accurate, consistent***, and ***testable***.
- Provides confidence through objective evidence that reliability, safety and quality goals will be met.

- Other objectives of performing V&V are to:
  1) Facilitate *early* detection and correction of software errors;
  2) Enhance management *insight into process and product risk*; and
  3) Support the software life cycle processes to ensure compliance with program *performance, schedule, and budget* requirements.

Measuring Economic Value



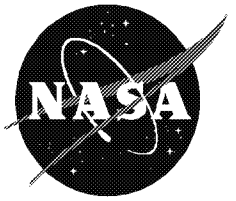
Financial and accounting measures report short-term and historical results. They fail to communicate whether an organization is creating economic value through…

- process improvements
- technological product innovations
- investments in tools, methods, and models

The Balanced Scorecard provides meaningful measures of the dynamics among organizational entities, their relationships and behaviors in achieving NASA IV&V mission through the use of...

- cause and effect graphing
- identifying performance and outcome measures
- linking these measures to financial performance

Balanced Score Card



NASA IV&V
Balanced Score Card

Strategic Goals: "To become an international leading force in the field of software engineering for improving: safety, quality, reliability, cost performance of software systems."

Finance

"How can we reduce costs and not compromise our mission?"

Customer

"To achieve our goals how do we want our customers to perceive us?"

Core Business Processes

"To satisfy our customers what business processes do we need to excel at to differentiate us and create a COE?"

Learning & Growth

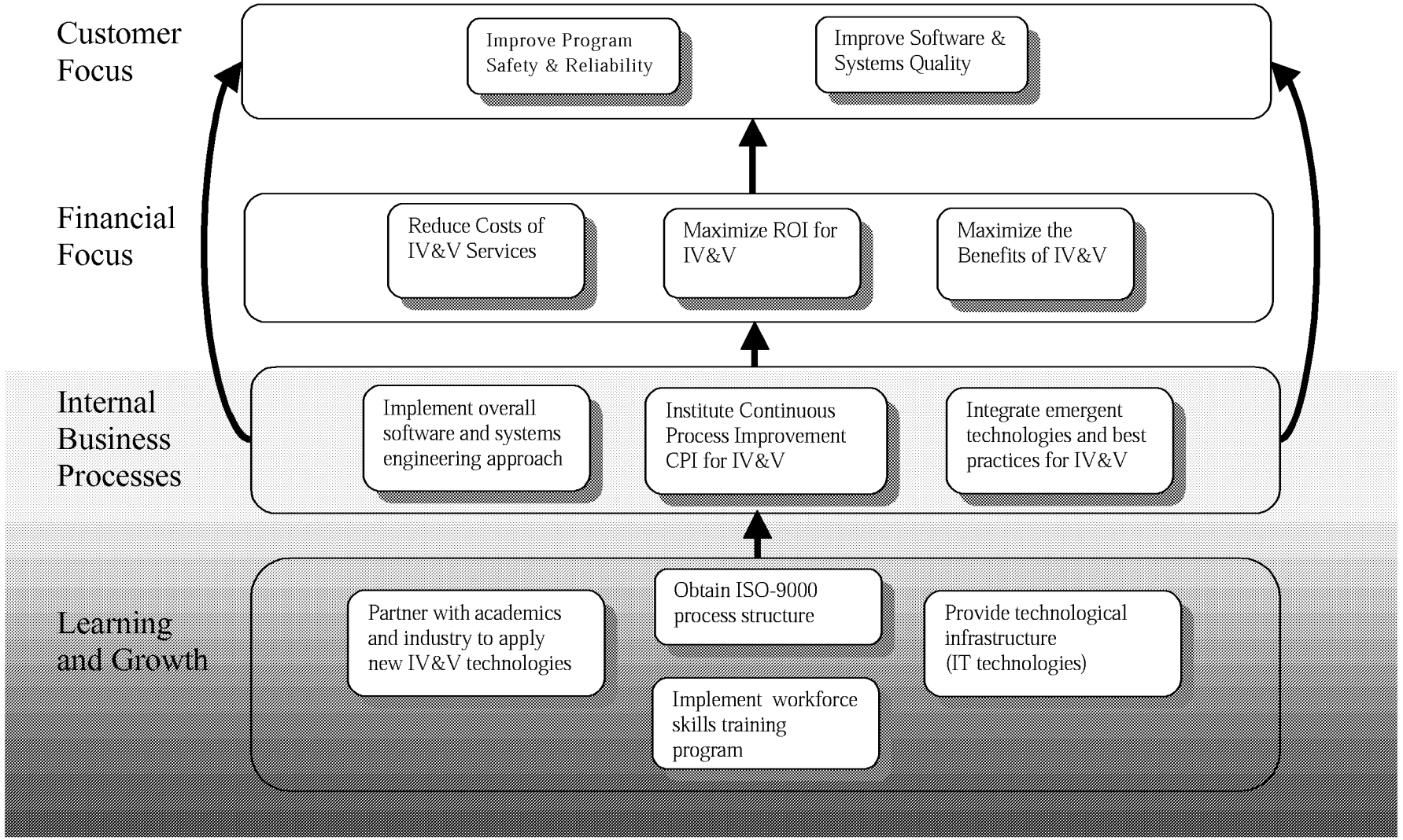
"What infrastructure do we need to sustain our ability to change and improve?"

ROI Research will provide objective measures for evaluating:

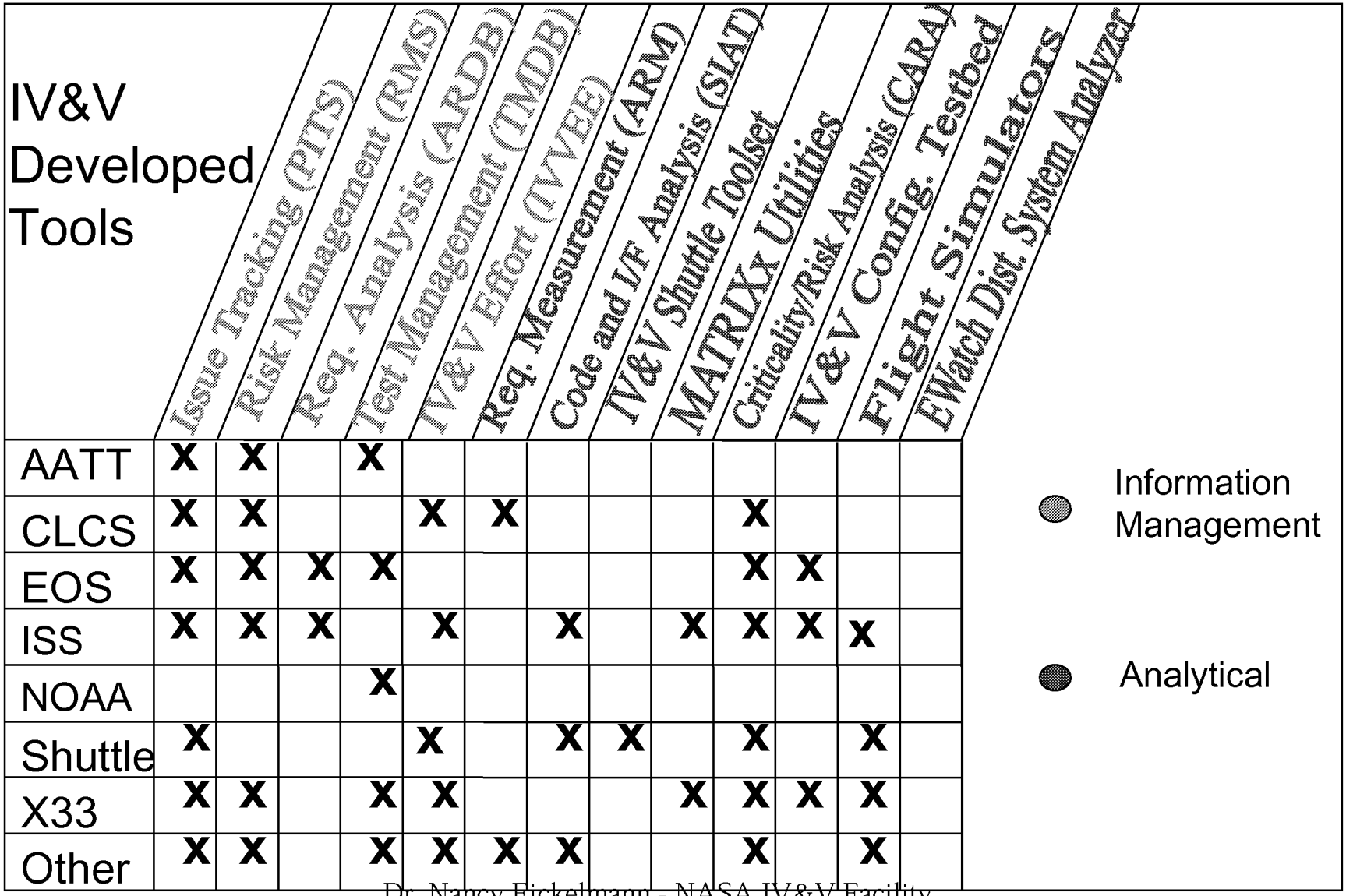
- **safety**
- **quality**
- **reliability**
- **cost**

NOTE: A Balanced ScoreCard (BSC) is a strategic measurement framework that makes visible the critical linkages between strategic goals and operational realities. BSCwas developed by Kaplan and Norton (Harvard Business School). Dr. Eickelmann has customized the framework for application in the NASA IV&V organization.

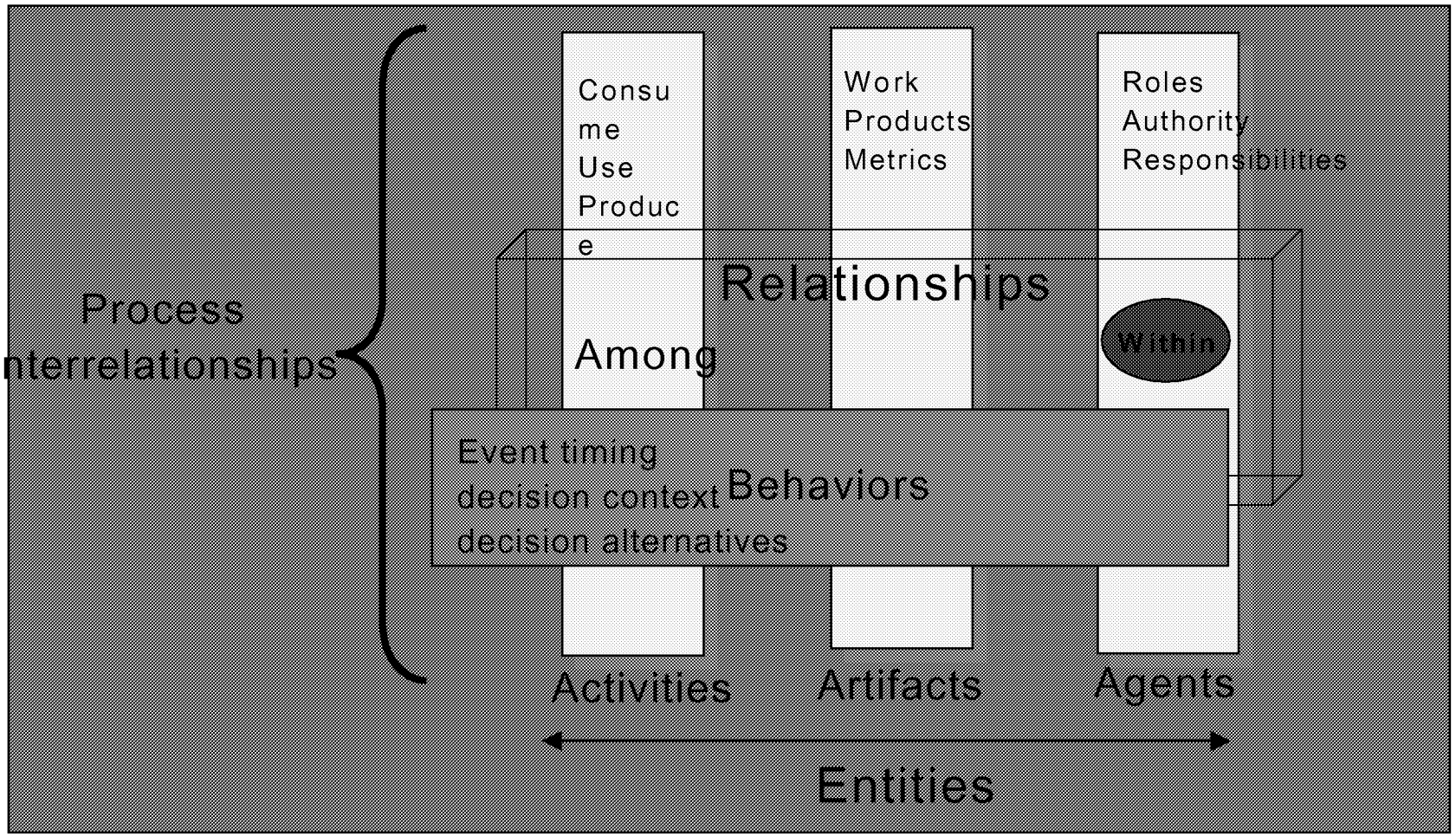
NASA IV&V
BSC Objectives



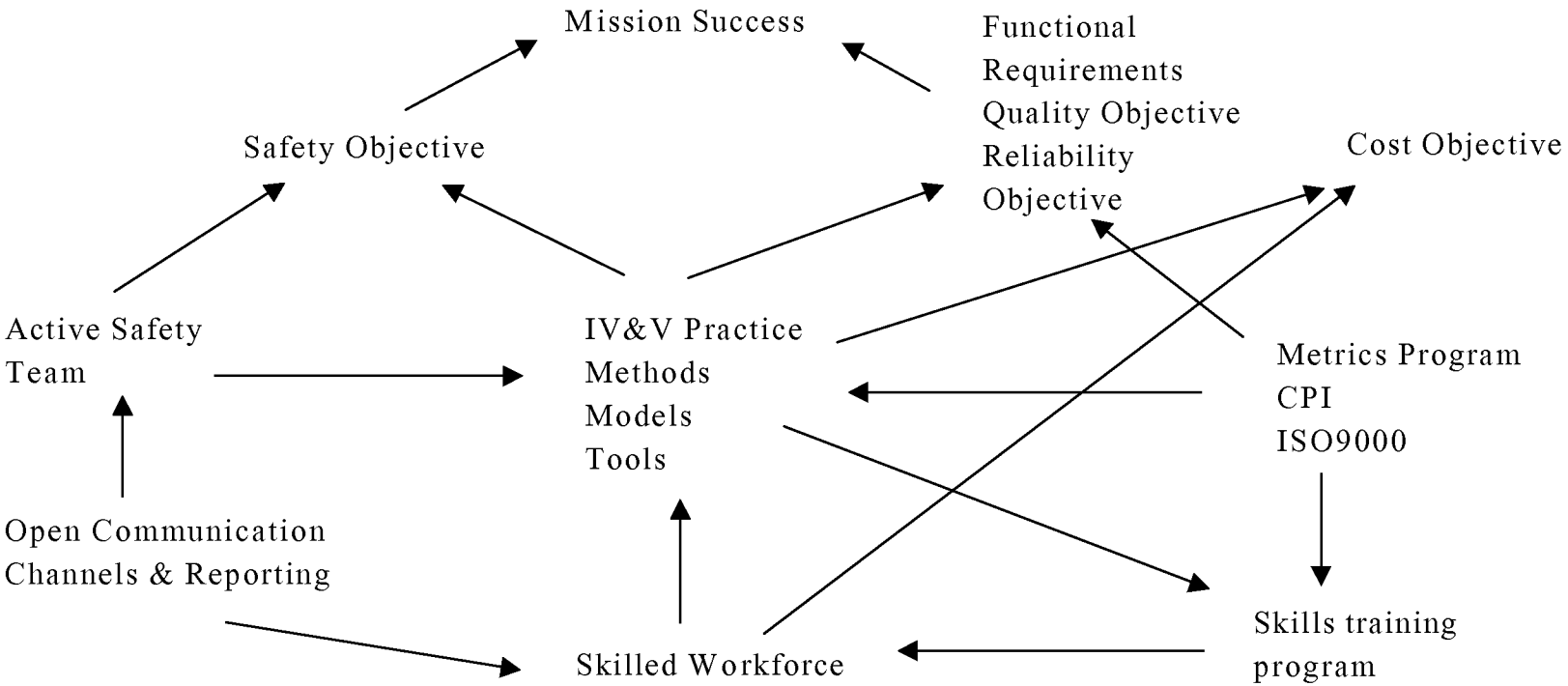
Commonly Applied IV&V Developed and COTS Tools



Process Definition Document
Metric Identification



Influence Diagram of Cause-Effect Graphing



NASA IV&V
BSC Measures

BSC Financial Measures	
Leading Indicators	IT - NPV & IRR
Lagging Indicators	ROI, ROM, ROA Cost reductions Value added

BSC Customer Measures	
Leading Indicators	Quality of service IV&V Responsiveness Surveys Benefit expectations
Lagging Indicators	# of Internal and External IV&V contracts (cross service)

NASA IV&V Strategic Goal Measures	
Safety	In-Flight Anomalies Post release defects
Quality	Severity 1 Defects
Reliability	MTTF,MTTR, MTBF
Performance	IT - Lag time
Cost	ROI, ROM, ROA Cost reductions Value added-NPV

BSC Infrastructure Measures	
Leading Indicators	Information dissemination lag time Communications effectiveness R&D project penetration % R&D technology transfer ratio
Lagging Indicators	Survey of customer satisfaction Appropriate staffing levels Strategic job coverage Strategic information availability Staff climate survey

BSC Core Process Measures	
Leading Indicators	Activity based costing Issue resolution cost Technology utility IT cost-benefit profiles
Lagging Indicators	TTEF–NPV+ IRR

Mapping IV&V Activities to BSC

Internal Business Processes	Objectives	Measures	Targets	Initiatives
	Minimize test effort	# test cases test pass/fail rates	Minimum set	Test Effectiveness
	Track issues to disposition	# Total Issues Open/Closed	Desired rate of closure	Web-based tracking system

IV&V Developed Tools

	Issue Tracking (PITS)	Risk Management (RMS)	Req. Analysis (ARM)	Test Management (TMDB)	IV&V Effort (IVVEE)	Req. Measurement (ARM)	Code and UF Analysis (SAT)	IV&V Shuttle Toolset	MATRIXx Utilities	Criticality/Risk Analysis (CARA)	IV&V Config. Testbed	Flight Simulators	Hybrid Dist. System Analyzer
AATT	x	x		x									
CLCS	x	x			x	x				x			
EOS	x	x	x	x						x	x		
ISS	x	x	x		x		x		x	x	x	x	
NOAA				x									
Shuttle	x				x		x	x		x		x	
X33	x	x		x	x				x	x	x	x	
Other	x	x		x	x	x	x			x		x	

Information Management

Analytical

Learning and Growth	Objectives	Measures	Targets	Initiatives
	Effective Communications	Reporting and response times	X+2(Y+!)	Web-based issue tracking
	World Class Research Staff	PhDs Publications	Top graduates Best in Class	Recruiting and Retention
	State of the Art Skills	Ongoing Training and certificates	100%	NASA HQ training program

Balanced Scorecard Perspectives

Information Sciences & Technology
Ames Research Center
ISO 9001
DNV Certification, Inc.
REGISTERED FIRM

	Objectives	Measures	Targets	Initiatives
Customers (Internal External)	No Losses	# Severity 1 &2	Remove < FRR	Formal Methods
	Reduce Risk	# IFA's	No Severity 1	Risk Management
	Manage Risk	Fault tolerance	Performance	Risk Mitigation

	Objectives	Measures	Targets	Initiatives
Financial	High ROI	Loss of Shuttle	> 100:1	ROI-CSIP
	Reduce Costs	Program costs	10% per FY	

	Objectives	Measures	Targets	Initiatives
Internal Business Processes	Minimize test effort	# test cases	Maximum Coverage	Test Effectiveness
	Track issues to disposition	# Total Issues Open/Closed	Rate of closure	Web-based tracking system

	Objectives	Measures	Targets	Initiatives
Learning and Growth	Skilled workforce	Degreed professionals	100%	Retention Strategies
	World Class Research Staff	PhDs Publications	Top graduates Best in Class	Recruiting and Retention
	State of the Art Skills	Ongoing Training and certificates	100%	NASA HQ training program

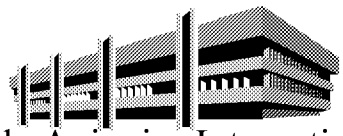
Space Shuttle Program Software Elements

Information Sciences & Technology

Ames Research Center

ISO 9001 DNV Certification, Inc. REGISTERED FIRM

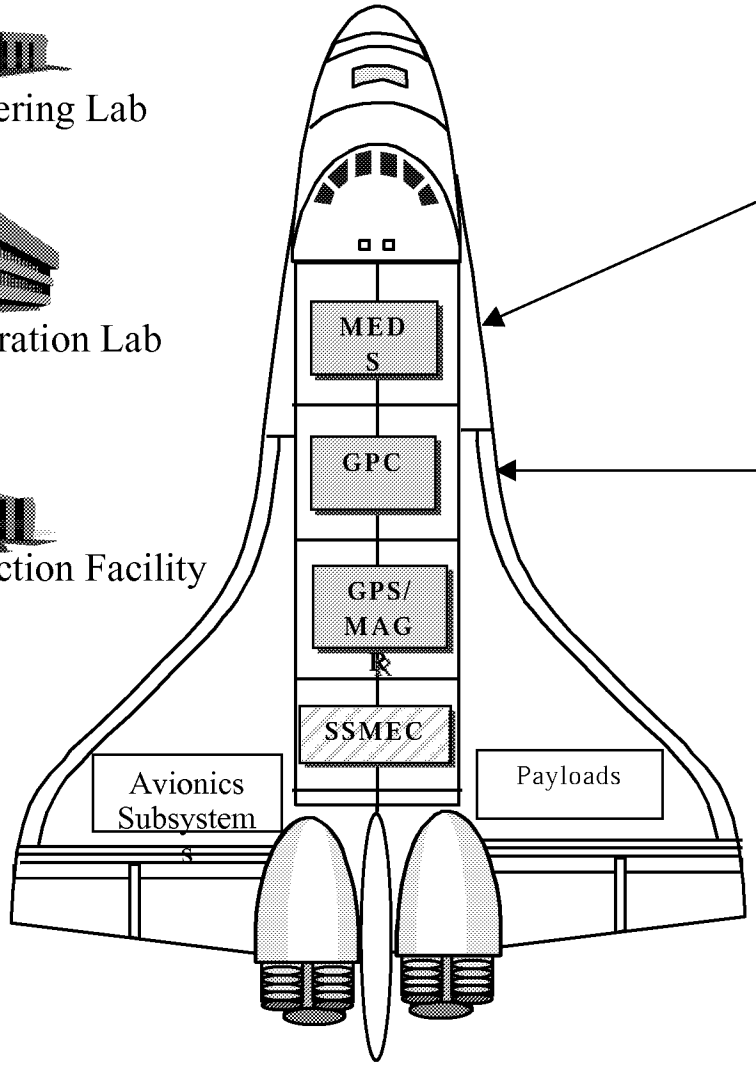
JSC Avionics Engineering Lab



Shuttle Avionics Integration Lab

SW Development/ Production Facility

Space Shuttle Vehicle



Launch Preparation

LPS

GSE

CLCS

System Software

Appl SW

Mission Control

Dolilu

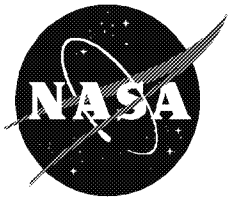
Mission Planning

Mission Training

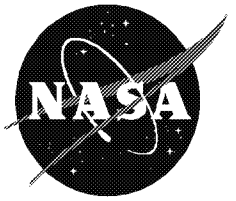
IV&V Activities

- **Past IV&V**
- **No IV&V**
- **IV&V**
- **Partial IV&V**

Summary



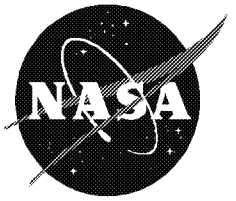
- BSC is a ***strategic measurement framework*** that NASA IV&V is applying to track strategic goals relative to operational activities for the Space Shuttle program.
- BSC ***measures economic value*** of operational activities relative to strategic goals
- BSC ***minimizes metrics required*** to evaluate product quality, safety, reliability, and cost.



Software Measurement Frameworks to Assess the Value of Independent Verification and Validation

Dr. Nancy S. Eickelmann
WVU/Software Research Laboratory
NASA Independent Verification and Validation Facility
Phone: (304) 367-8444
E-mail: Nancy.Eickelmann@ivv.nasa.gov
Website: http://research.ivv.nasa.gov/~ike
December 1999

Motivation
Research Roadblocks and Directions



ROI Directions

Clinger-Cohen Act of 1996 and the Government Performance and Results Act (GPRA) mandates the use of …

> "a framework of modern technology management based on practices followed by leading private-sector and public-sector organizations that have successfully used technology to dramatically improve performance and meet strategic goals." (Source GAO/NSIAD-98-181).

ROI Roadblocks

Four significant barriers to measuring financial performance related to information technologies included:

- Difficulty of measuring economic benefits
- Inability to determine returns
- Lack of good metrics
- Incomplete records/accounting of investments

ROI Research Initiative applies the strategic measurement framework developed at the Harvard Business School known as the Balanced Score Card BSC.

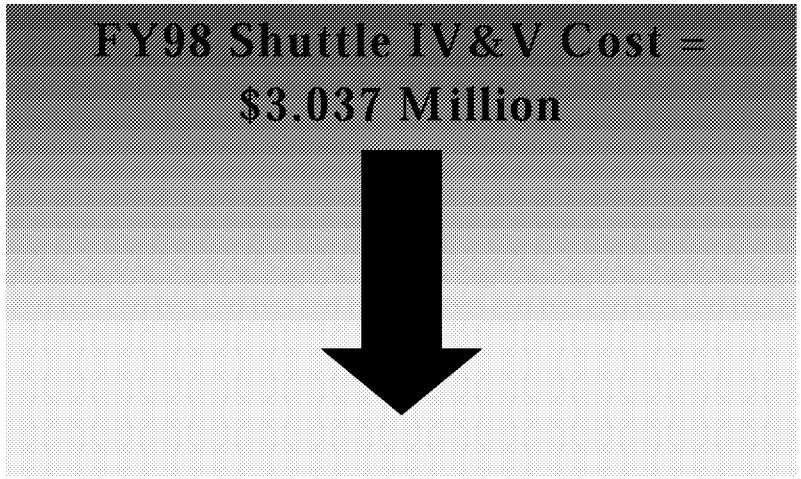
NRC Based Financial Measures
1999 Space Shuttle ROI

Information Sciences & Technology
Ames Research Center
ISO 9001
DNV Certification, Inc.
REGISTERED FIRM

Severity 1 & 2 Issue Tracking Report flight software.

Shuttle Cost Risks (Severity 1)
- Loss of life = Priceless
- Loss of Shuttle = *$2Billion**
- Loss of hardware = $?$
- Causes a Stand-down = $?$

Shuttle Cost Risks (Severity 2)
- Mission objectives not met
- Manifest redeployment based on Standard Flight Cost $?$

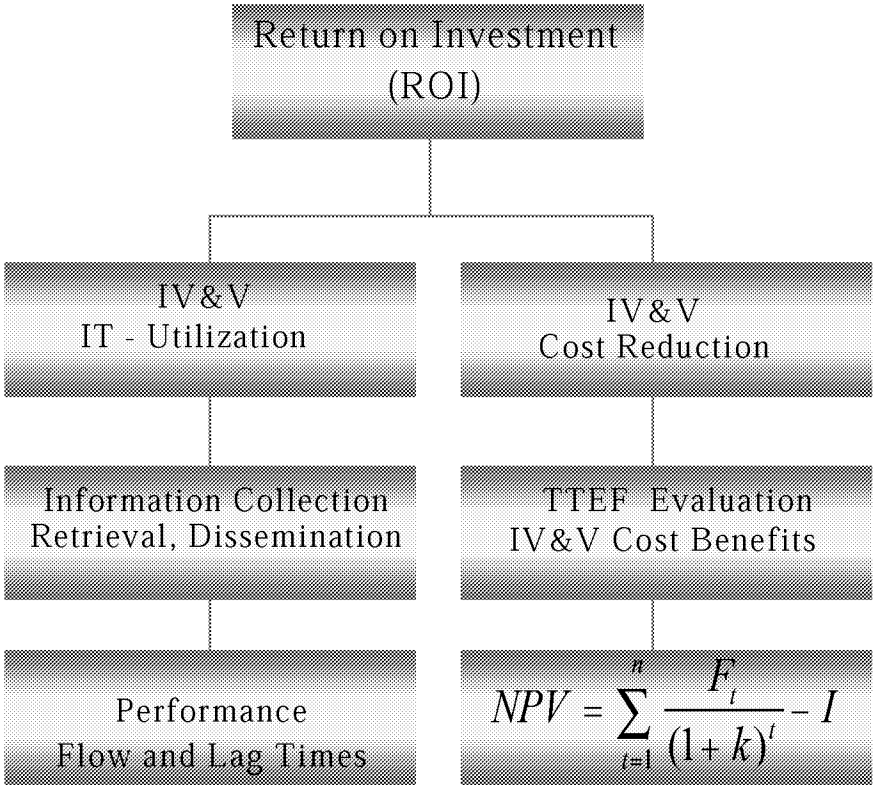


ROI = 863:1

1) FY98 Shuttle IV&V costs: $3.037M
2) Average Shuttle mission cost: $409M for FY98 (was $590M in FY93); taken from FY2000 budget package chart

*Amount shown is in 1993 dollars and was reported by the National Research Council

NASA IV&V Linking from Financial - Process - Infrastructure Measures



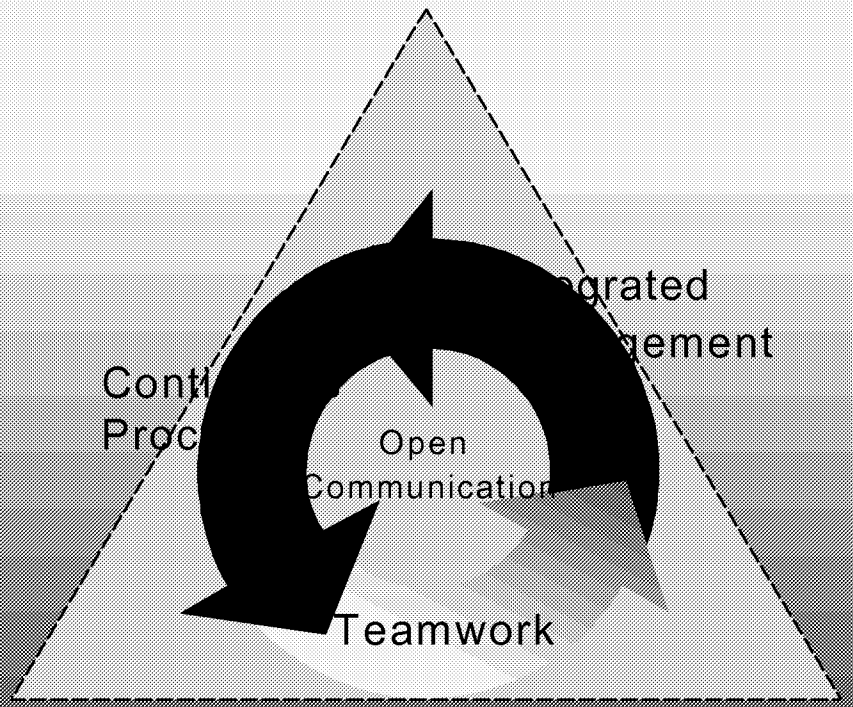
The BSC strategic measurement framework links cost benefit analysis to evaluating the effectiveness of technologies in achieving strategic goals.

Internal Process Measures for IV&V as Risk Management

IV&V brings a unique contribution with respect to Risk Management.

The core principle of risk management is ***open communication*** that enables:

- free-flowing information between all project and organizational levels
- facilitating formal, informal, impromptu communication
- applying consensus and valuing individual contributions



Source: Continuous Risk Management Guidebook

Space Shuttle Product Lines Process Reduces Risk

Information Sciences & Technology
ISO 9001
DNV
DNV Certification, Inc.
REGISTERED FIRM
Ames Research Center

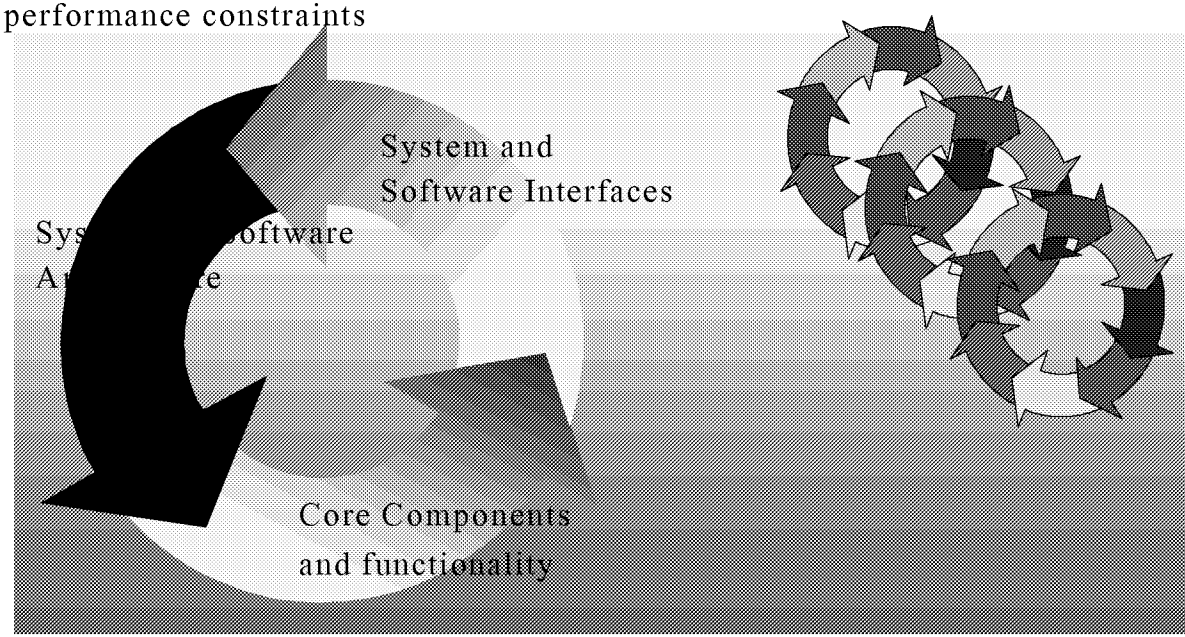
NASA

Domain Engineering for the Space Shuttle Product Line

•Requirements Engineering for every OI
•Architectural Integrity through negotiated modifications
•Maintained extensibility and modifiability of design through restriction of violations of architectural structural, functional, and performance constraints

Application Engineering

•Reuse of core assets for every OI
•Architectural Integrity through modularity and encapsulation
•Maintained extensibility and modifiability of design



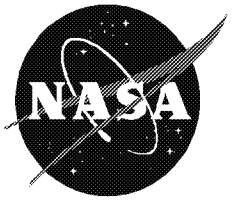
Multiple Operational Increments are developed using a core set of software CSCIs.

The core functionality is modified to accommodate specific mission requirements relative to launch and load.

A primary product line benefit is a result of test environment reuse. Test suites are regressed and reused and test plans are readily extensible to V&V system and software modifications to core assets.

Dr. Nancy Eickelmann - NASA IV&V Facility

IV&V Challenges

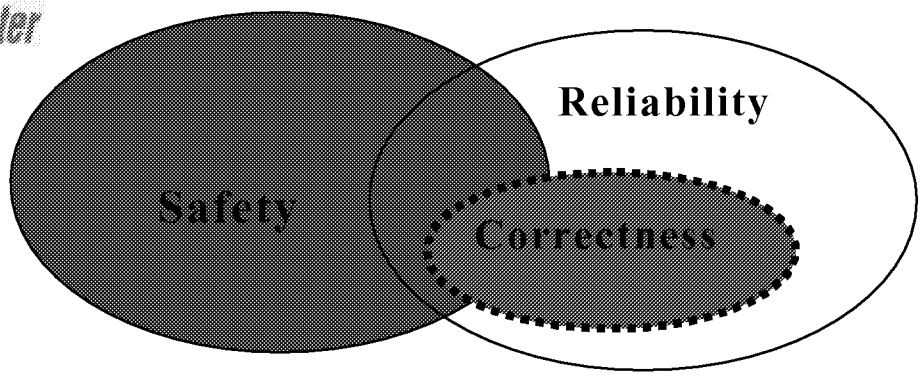


Challenges

- Software Modularity
- COTS
- Program Mngt
- Software Engineering
- SW Reuse
- Maintainability
- Fail Safe
- Risk Control

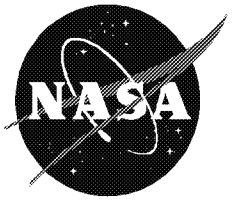
- Technology Transfer
- Configuration Management
- Process IV&V across Series
- Validation of Flight Software
- Configuration Management for all developed elements
- Establish Program-wide Standards

Customer Themes for Mission Success
Safety-Reliability-Quality



- The relationships among these customer themes are significant as they are not independent of one another and therefore must be analyzed based on their degree of covariance and interaction.
- Safety requires that unsafe states cannot be entered from any point of function of the system.
- It is possible for the systems to function reliably that is without failure and still enter unsafe states of operation.
- A system can be completely correct and defect free and still enter unsafe states.

Example Customer Measures ROI



Objectives	Sample Measures
Customer partnership and involvement	*percent projects using integrated project teams* *percent joint IT customer/supplier service lever agreements*
Customer satisfaction	*percent customers satisfied with IT product delivery* *percent customers satisfied with IT problem resolution* *percent customers satisfied with IT maintenance and support* *percent customers satisfied with IT training* *percent products launched on time* *percent service-level agreements met*
Business process support	*percent IT solutions supporting process improvement projects* *percent users covered by training to use new IT solutions* *percent new users able to use applications unaided after initial training*

*GAO /AIMD-97-163

Example Financial Measures and Strategic IT Goals

Objectives	Sample Measures
Enterprise mission goals	*percent mission improvements (cost, time, quality, quantity) attributable to IT solutions and services* *percent planned IT benefits projected v. realized*
Portfolio analysis and management	*percent IT portfolio reviewed and disposed* *percent old applications retired* *percent applications retirement plan achieved* *percent reusability of core application modules* *percent new IT investment v. total IT spending*
Financial and investment performance	*percent and cost of services provided in-house v. industry standard* *IT budget as percent of operational budger and compared to industry average* *pet present value, internal rate of return, return on investment, return on net assets*
IT resource usage	*percent consolidated/shared resources across units* *percent cross-unit shared databases and applications* *percent hardware/software with interoperability capabilities*

*GAO /AIMD-97-163

Example Business Process Measures - ROI

Objectives	Sample Measures
Applications development and maintenance	*number of function points delivered per labor hour* *number of defects per 100 function points at user acceptance* *number of critical defects per 100 function points in production* *percent decrease in application software failures, problems* *mean time to resolve critical defects* *cycle time for development*
Project performance	*percent projects on time, on budget* *percent projects meeting functionality requirements* *percent projects using standard methodology for systems analysis and design*
Infrastructure availability	*percent computer availability* *percent communications availability* *percent applications availability* *on-line system availability*
Enterprise architecture standards compliance	*number of variations from standards detected by review and audit per year* *percent increase in systems using architecture* *percent staff trained in relevant standards*

*GAO /AIMD-97-163

Example Infrastructure Measures ROI

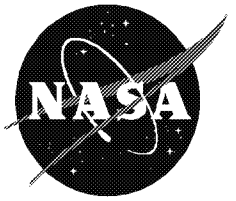
Objectives	Sample Measures
Workforce competency and development	*percent staff trained in use of new technologies and techniques* *percent staff professionally certified* *percent IT management staff trained in management skills* *percent IT budget devoted to training and staff development*
Advanced technology use	*percent employees skilled in advanced technology applications* *number of dollars available to support advanced technology skill development*
Methodology currency	*currency of application development methods in used* *percent employees skilled in advanced application development methods* *percent projects developed using recognized methods and tools*
Employee satisfaction and retention	*percent employee satisfaction with the capability of the existing technical and operating environment to support mission* *percent employee turnover by function*

*GAO /AIMD-97-163

IT - BSC Measures

1. Identify achievable IT goals by NASA Center
2. Classify goal structure
   - Short-term
   - Medium-term
   - Long-term
3. Categorize the impact of proposed IT strategies
   - Very high impact
   - High impact
   - Low impact
   - Very low impact
4. Estimate probability of IT success (0 to 1)
   - 10% up to 100%
5. Identify appropriate IT measures for each NASA Center based on ***their*** strategic goals and mission

TTEF



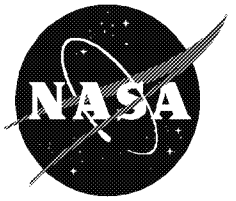
test effectiveness based on a relative test detection rate and test failure rate given the optimistic or pessimistic inaccuracy of a specific test strategy

test productivity based on the rate of execution and verification of successful test inputs with a test output that results in an observed failure or detection

test schedule compression based on full life cycle analysis

It then relates these factors in a properly scoped cost benefit analysis methodology applying ROI, NPV and IRR.

TTEF - NPV - IRR

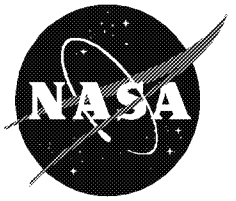


The TTEF evaluates the degree of achievement of the primary goals including test effectiveness, test productivity, and test schedule compression using a comparative analysis...

$$NPV = \sum_{t=1}^{n} \frac{F_t}{(1+k)^t} - I$$

...and is designed to capture the principle benefits associated with a technology as well as the complete cost structure that has been extended to use NPV and IRR.

TTEF Example

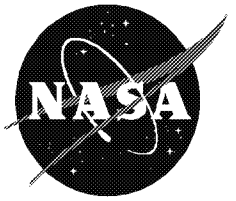


- A specific example is chosen that uses commercially available test tools and object-oriented modeling and specification documentation tools.

  "The availability of these already integrated tools enabled the team to measure the costs and benefits of reusing object models for testing without having to deal with the sometimes confusing cost and productivity factors..." Communications of the ACM, September 1994/Vol..37, No. 9.

- A comparison is conducted between current practice for evaluating technologies and our Test Technology Evaluation Framework (TTEF).

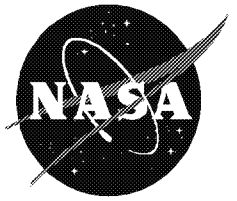
Naïve Cost Benefit Analysis



$$ROI = \frac{152 hours}{0.5 hours}$$

- Industry average test case productivity (Capers Jones) 20 to 300 test cases per month
- COCOMO Man Month is equal to 152 hours
- Cost of model development 24 minutes
- Test benefit with automated OMT 151.5 hours saved

Cost Shifting Versus Cost Savings



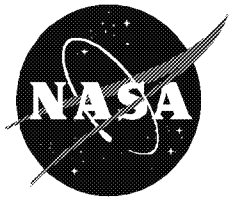
StP/OMT Claims

- Cost savings (benefit) of one person month.
- Cost of model production 24 minutes.
- ROI of 304:1

Reality

- Cost shifting to model development and evaluation of model fidelity.
- Cost shifting to tools, training, updates, reuse of models.
- Test effectiveness was not evaluated...

Initial Costs



Poston's Case study used OMT technologies and evaluated automated specification-based testing technologies:

- StP/OMT (Integrated Development Environments, IDE) - Development tool based on James Rambaugh's object modeling technology was used to develop the specification Single license cost: $6,500. Plus one year maintenance cost $1,300.

- StP/T (Integrated Development Environments, IDE) - Test case generation tool based on OMT specifications was used to generate the test cases from the specification Single license cost: $10,530 plus $1,580 maintenance.

- XRUNNER - Software test tool test execution, capture replay was used to run the test cases. Single license costs: $9,503 plus $2,100 annual maintenance fee.

OMT Lifecycle Savings

OMT LIFECYCLE

Analyze Problem Write Requirements	Design System	Design Objects	Implement Objects	Debug and rework requirements, designs, programs, tests		
				Define-Design-Write Test Cases	Execute Test Cases	Evaluate Tests and Software

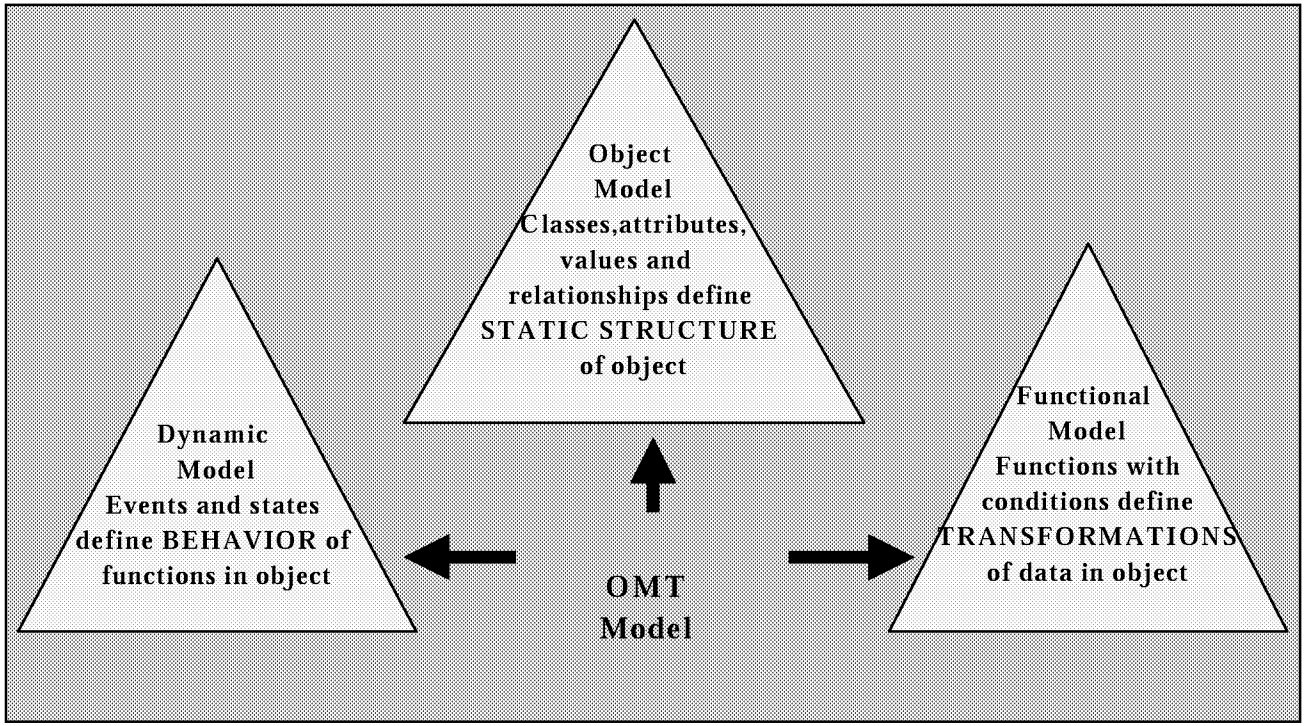
OMT LIFECYCLE with AUTOMATED TEST

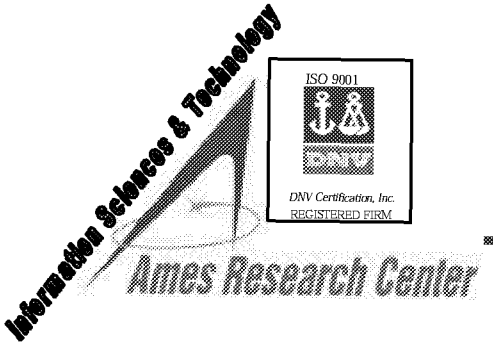
Analyze Problem Write Requirements	Design System	Design Objects	Design System	Debug and rework requirements, designs, programs, tests		Time Savings
	Define-Design-Write Test Cases			Run Test Cases	Evaluate Test Cases	

Communications of the ACM, September 1994/ Vol.37, No. 9.

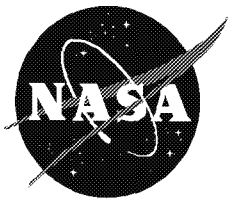
Automated OMT and Test Tools

Rumbaugh's OMT uses three separate views to graphically describe the system. OMT provides support to create models to represent system objects, dynamics and functional aspects. Evaluate benefits w.r.t. the technology window…





Scoping Test Cost Benefits Using TTEF



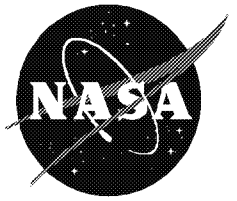
The software test process and test process automation include six canonical functions that are used to evaluate test technology impacts:

- test execution
- test development
- test failure analysis
- test measurement
- test management
- test planning

The software test technology impacts are evaluated for impacts in the following categories in the context of test functionality and lifecycle context:

- test effectiveness
- test productivity
- test schedule compression

Test Automation Cost

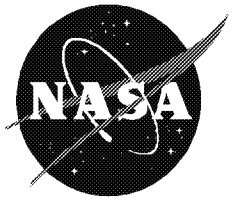


• The costs associated with adopting Poston's technology (or similar automated testing tools) are associated with the paradigm shift to OO technology and anticipated reuse benefits w.r.t. the technology window:

- Equipment purchase StP/OMT, StP/T, Xrunner*
- Reuse libraries*
- Training of personnel*
- OMT Specification development and analysis
- Verification of OMT model fidelity
- Maintenance and archiving of reusable assets
- Software updates

** Initial costs in adopting test automation technologies.*

Test Automation Benefits

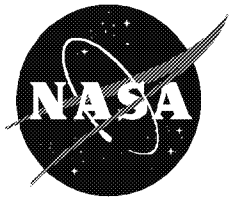


• The benefits associated with adopting Poston's technology (or similar automated testing tools) are associated with the paradigm shift to OO technology and anticipated reuse benefits w.r.t. the technology window:

- Test personnel time savings
- Automated test case generation and execution
- Test schedule savings
- Design model reuse*
- Test specification reuse*

** Benefits may accrue to future projects based on reuse savings.*

NPV Calculations



$$NPV = \sum_{t=1}^{1} \left(\frac{152.5 hours \times labor\ rate_1}{(1+k)^1} \right) - 30 minutes \times labor\ rate$$

Naïve NPV = $2,763.00

Full Costs NPV = $(20,854.00)

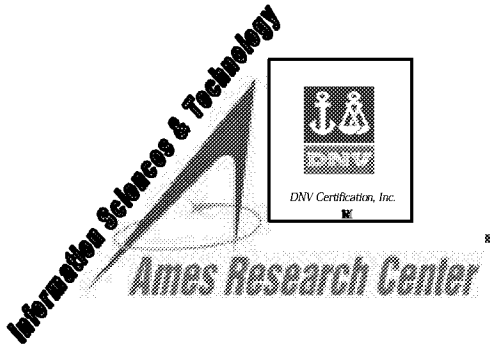
- Assumes one time period
- Applies only labor savings costs
- Labor rate - $20.00/hour
- Discount rate 10%

Assumes one time period

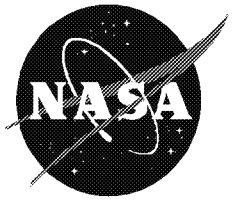
Applies capital ($20,036) and labor

Labor rate - $20.00/hour

Discount rate 10%



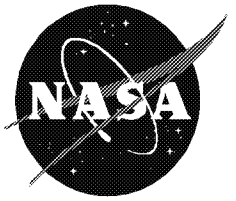
NPV Calculations



TTEF NPV = $1819.00 (in year one with reuse over 10 projects)

- Automated specification based testing initial cost estimate - $20,036.00 per seat
- Annual maintenance costs - $ 3,680.00
- Additional savings through reuse strategies - $2772.00 per additional program (example uses 10 programs)
- Discount rate 10%
- *Improved test effectiveness savings - difference between old rate and new, not calculated**

Cost-Benefit Analysis



- A case study was performed to evaluate the measurement framework that provides guidelines for how the data will be interpreted, this work has been accepted for publication the results demonstrate:
- The value of the technology was significantly overstated when the scope of the cost-benefit analysis was underspecified, ROI 304:1.
- The value of the technology was understated when the scope was applied correctly but the underlying reuse process was not taken into account, ROI 0.1517:1.
- Finally when both the scope of the cost-benefit analysis and the underlying process and reuse paradigm are incorporated into the analysis meaningful quantification of the value of the technology results.

Session 4: Space Software

Lou Blazy, NASA/Ames/WV

Marti Szczur, NASA/Goddard

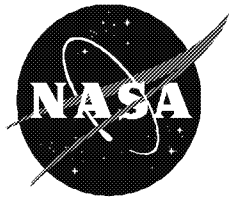
Richard Doyle, JPL

Dr. Louis J. Blazy
Director, IV&V Facility, Software Technology Division
Information Sciences and Technology Directorate
Ames Research Center

December 1, 1999

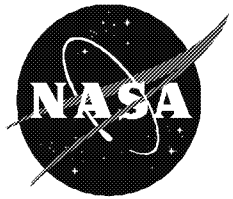
Phone: (304) 367-8202
E-mail: Louis.Blazy@ivv.nasa.gov
Website: http://www.ivv.nasa.gov

Mission



- To be world-class creators and facilitators of innovative, intelligent, high performance, reliable information technologies that enable NASA missions.
  - Increase software safety and quality through error avoidance, early detection and resolution of errors, by utilizing and applying empirically based software engineering best practices.
  - Ensure customer software risks are identified and/or that requirements are met and/or exceeded.
  - Research, develop, apply, verify, and publish software technologies for competitive advantage and the advancement of science.
  - Facilitate the transfer of science and engineering data, methods, and practices to NASA, educational institutions, state agencies, and commercial organizations.

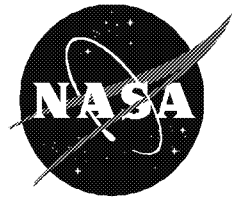
Goals



- To become a national Center Of Excellence (COE) in software and system independent verification and validation.

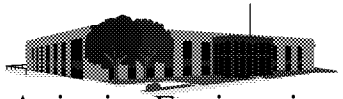
- To become an international leading force in the field of software engineering for improving the safety, quality, reliability, and cost performance of software systems.

What are the Problems we are Trying to Solve?

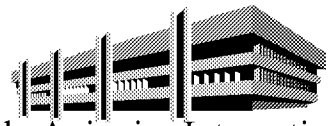


- Ensure safety of NASA missions
- Ensure requirements are met
- Minimize programmatic and technological risks of software development and operations
- Improve software quality
- Reduce costs and time to delivery
- Improve the science of software engineering

Space Shuttle Program Software Elements

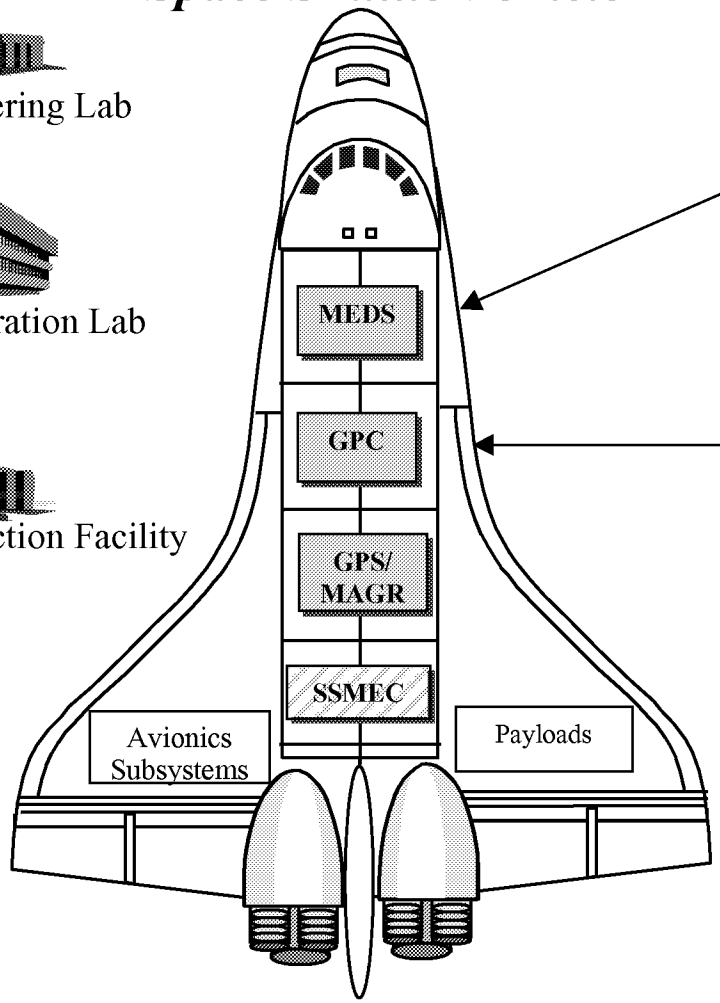


JSC Avionics Engineering Lab



Shuttle Avionics Integration Lab

SW Development/ Production Facility



Launch Preparation

LPS

GSE

CLCS

System Software

Appl SW

Mission Control

Dolilu

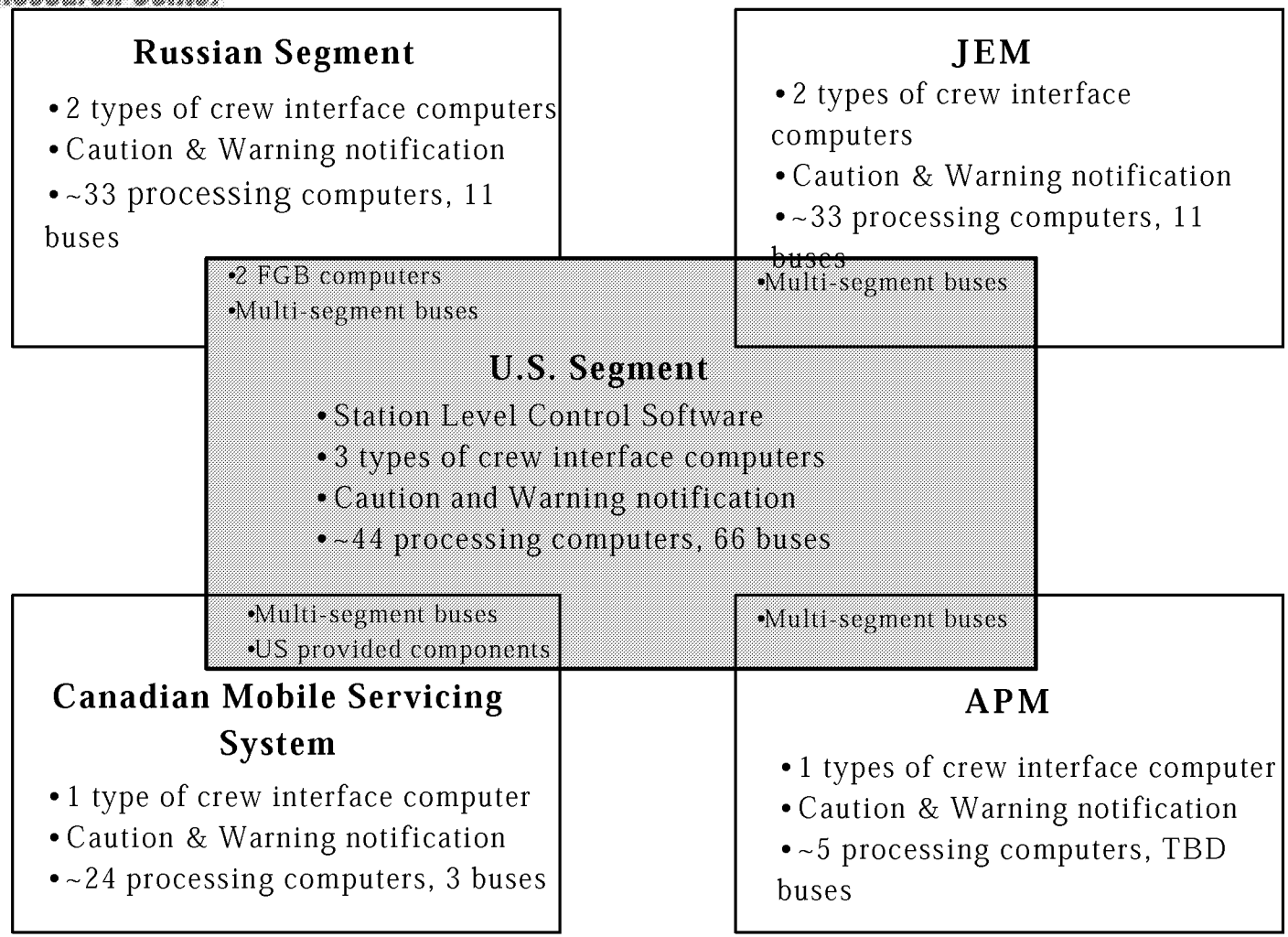
Mission Planning

Mission Training

IV&V Activities

- **Past IV&V**
- **No IV&V**
- **IV&V**
- **Partial IV&V**

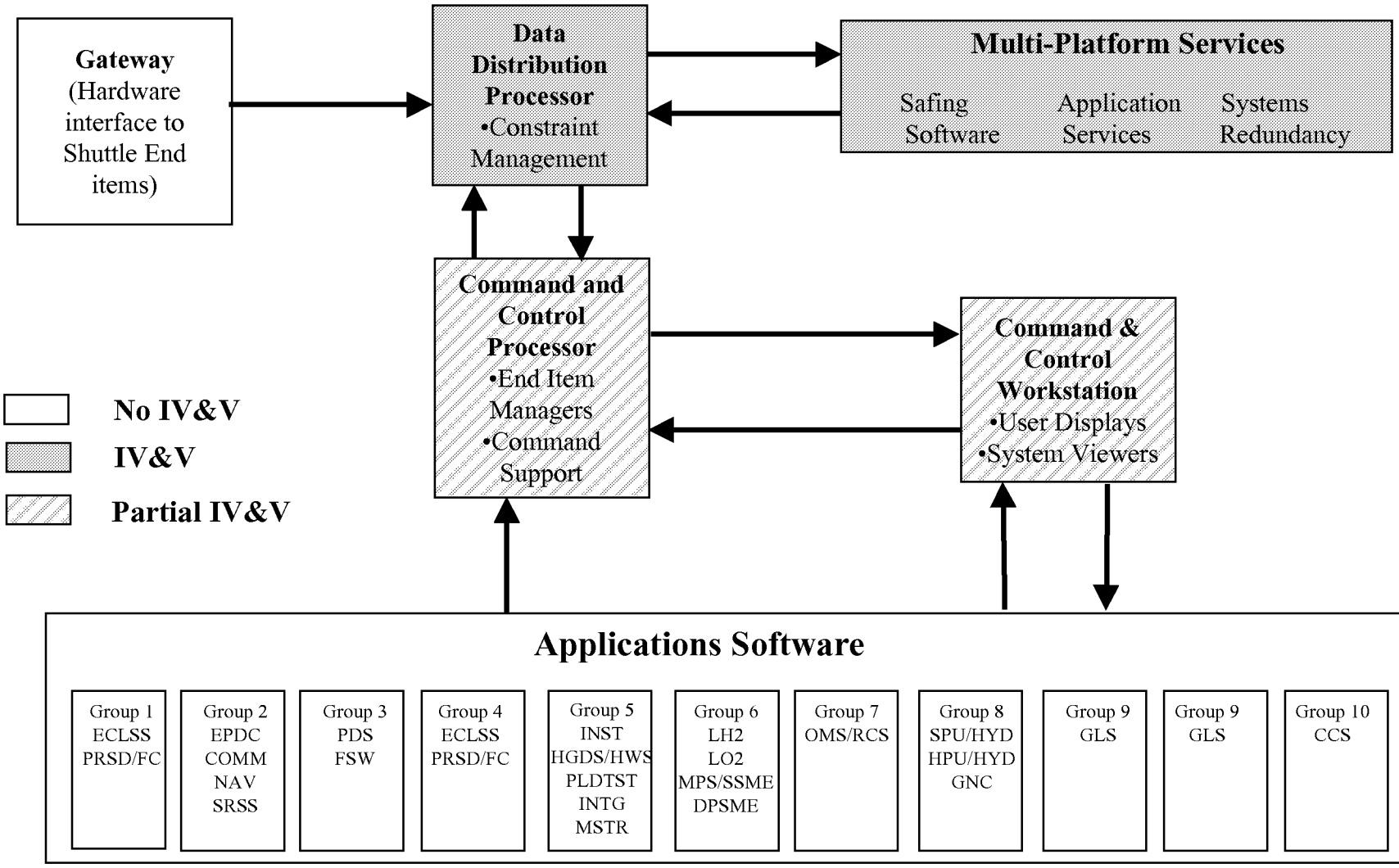
Station Computer Systems at Assembly Complete



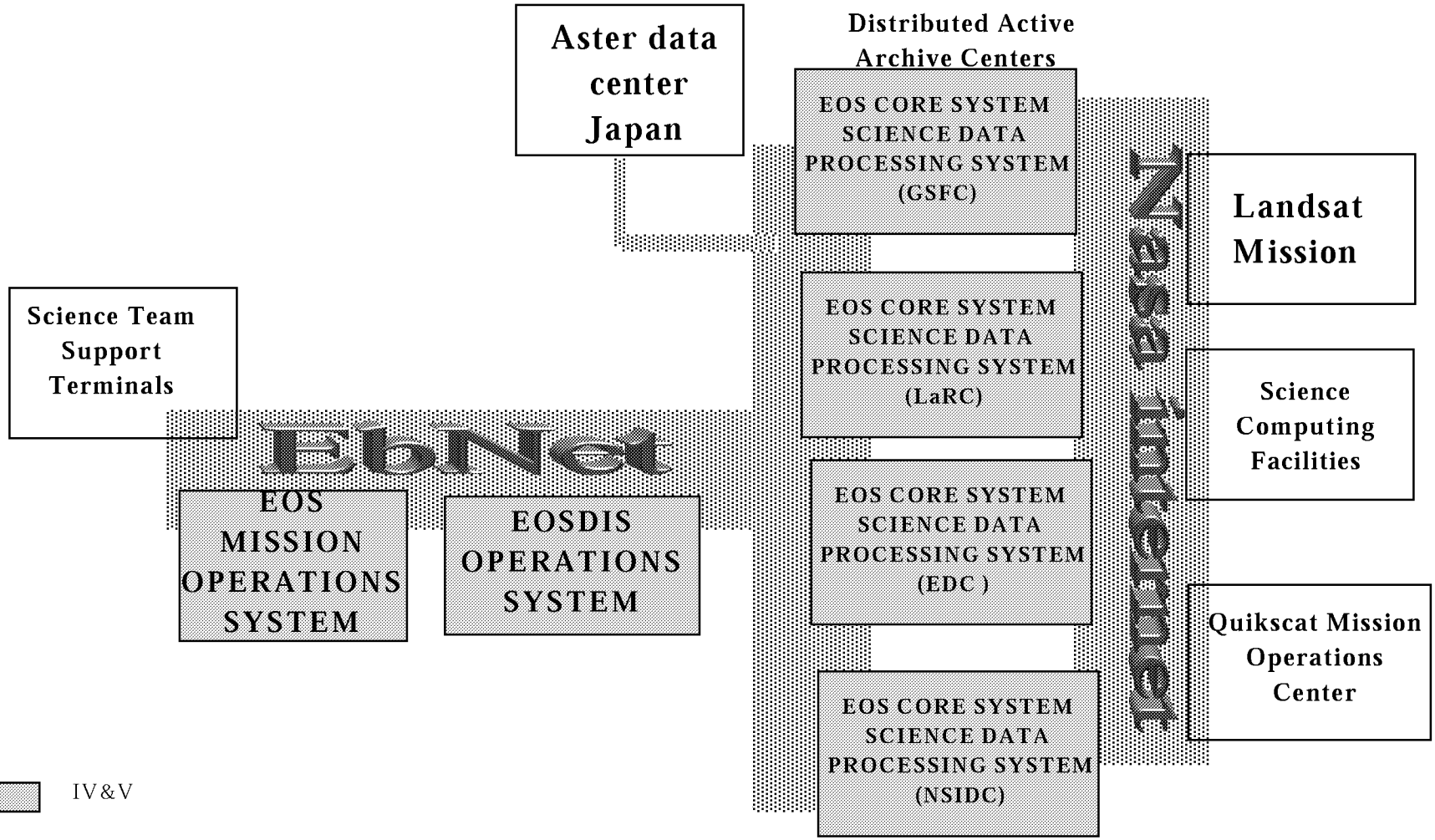
IV&V

No IV&V

CLCS System Architecture



EOSDIS
Systems Architecture



* reference EOS Ground system Architecture Concept (diagram), ESDIS ICWG
EOS Ground System contains approximately 30 component types, hundreds of components

Major IV&V Project Accomplishments

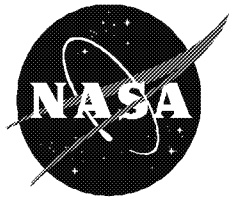
Accomplishments FY99

- Shuttle
  - Identified 15 software errors that could produce a loss of Shuttle or crew (e.g., shutdown of all SSME, loss of control during ascent)
- Station
  - Identified critical discrepancy in Mode Control software that rendered orbiter docking zero fault tolerant
- CLCS
  - Identify critical discrepancy with Command Management computer software that controlled the Hypergalic Maintenance Facility

Program Benefits

- Shuttle
  - Increased mission safety, reliability, and reduce cost
- Station
  - Enhanced reliability
- CLCS
  - Enhanced system reliability

Applied Software Technology Research Program

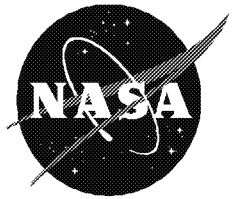


- **Software Metrics**
  - Return On Investment (ROI)
  - IV&V test method/practice effectiveness
  - Program effectiveness measures (schedule, cost, quality, productivity)

- **Software Reliability**
  - Curvilinear regression models
  - Diagnostic tools
  - Static code analysis

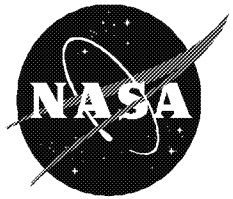
- **Software Risk Assessment**
  - Model Fidelity Assessment

Applied Software Technology Research Program



- **V&V for Intelligent Systems**
  - Autonomous spacecraft & rovers
  - Automated verification and validation
  - Predictive hazard analyses
- **V&V for Cost Estimating**
  - Predictive cost estimating
- **Software Requirements Engineering**
  - Abductive reasoning
  - Model based reasoning

Applied Software Technology Research Program



- **Specification-based Testing**
  - Automatic test case generation
  - Test coverage analysis
  - Empirically based formal verification

- **Software Architectural Analysis**
  - Safety, reliability, evolveability of different architectures
  - V&V of COTS

- **Software Re-use and the development of product families**
  - High assurance software
  - Domain engineering

FY00 Center Initiatives Research Program

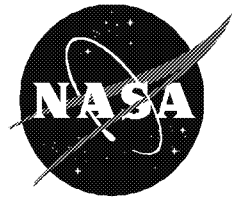
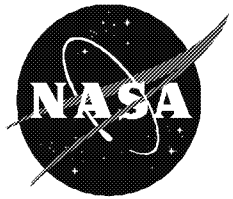


	Table 1. FY00 Center Initiatives
	Projects
	Ames Research Center (ARC)
1	Verification and Validation of Model Based Autonomous Systems
2	Specifying Planning, Execution and Monitoring behaviors for Autonomous Systems
3	The High Quality Knowledge Based Initiative
4	Automation of Testing, Monitoring, and Troubleshooting for Control Systems utilizing Automatic Code Generators
	IV&V Facility
5	University Research
6	Science and Engineering Technical Assessments
7	NASA Affiliate Membership in the Software Productivity Consortium
8	Development of Return on Investment (ROI) Model and Metrics for Software System Independent Verification and Validation
9	Safety Metrics for Human-Computer Controlled Systems
10	Investigation of the Engineering and Research Issues in Model-Based Verification for SW Systems
	Glenn Research Center (GRC)
11	Update of NASA Guidebook for Safety Critical Software Analysis and Development
12	Software Development Planning Tool with Cost Estimation and Risk Assessment
	Goddard Space Flight Center (GSFC)
13	Software Assurance Tools
14	Software Reliability Metrics
15	Intelligent Data Filtering
16	Software Reuse Frameworks for Flight Software
17	COTS Classification, Lifecycle management, Metrics, Reliability

FY00 Center Initiatives Research Program

	Table 1. FY00 Center Initiatives
	Projects
	Jet Propulsion Laboratory (JPL)
18	Reducing Software Security Risk through an Integrated Approach
19	Formalized Pilot Study of Safety Critical Software Anomalies - JPL
20	Advanced Risk Reduction Tool (ARRT)
	Johnson Space Center (JSC)
21	MicroElectro-Mechanical Systems (MEMS) Data Security for Human Exploration and Development of Space (HEDS)
22	Real-Time Embedded Software Verification and Validation
	Langley Research Center (LaRC)
23	Using Software Engineering Methods and Techniques to Improve the Simulation Environment
24	Software Process Improvement
	Marshall Space Flight Center (MSFC)
25	V&V of Embedded Real-Time COTS Operating Systems for Space Flight Projects
26	Improvement of Software Development Processes through an Innovative Software Process Improvement Methodology
	Stennis Space Center (SSC)

Research Accomplishments



Accomplishments

- Space Station
  - Discovered anomaly in requirements that specify behavior of Fault Detection, Isolation, and Recovery (FDIR) procedures in the 1553 Bus
- Space Shuttle
  - Discovered potential anomalies in Miniaturized Airborne GPS Receiver (MAGR S3S) software used for onboard GPS-based GN&C
- CLCS
  - Discovered anomaly in Redundancy management/Reliable messaging protocols that controls selection of Master-Slave workstation relationships
- Cassini - Mission to Saturn
  - Discovered anomaly in mark-and-rollback state table broadcast algorithm that controls error recovery procedures in onboard Command and Data Subsystem
- DFRC F-18 PSFCC V&V
  - Discovered anomaly in embedded flight software written in Ada programming language that controls dynamic flights surfaces on experimental F-18

Program Benefits

Space Station
Increased mission safety by initiating extensive redesign and simplification of Space Station main bus Fault Detection, Isolation, and Recovery procedures

Space Shuttle
Increased mission safety by identifying causes of current and potential lockups in MAGR GPS - Shuttle GPC link

CLCS
Enhanced design and robustness of reliable multicast protocol used in core CLCS data distribution scheme

Cassini - Mission to Saturn
Increased mission safety and viability by triggering redesign of core operating system fault recovery procedures during critical sequence execution

DFRC F-18 PSFCC V&V
Increased flight safety during experimental flights by increasing confidence in frame completion deadlines for real-time software

Goddard Space Flight Center

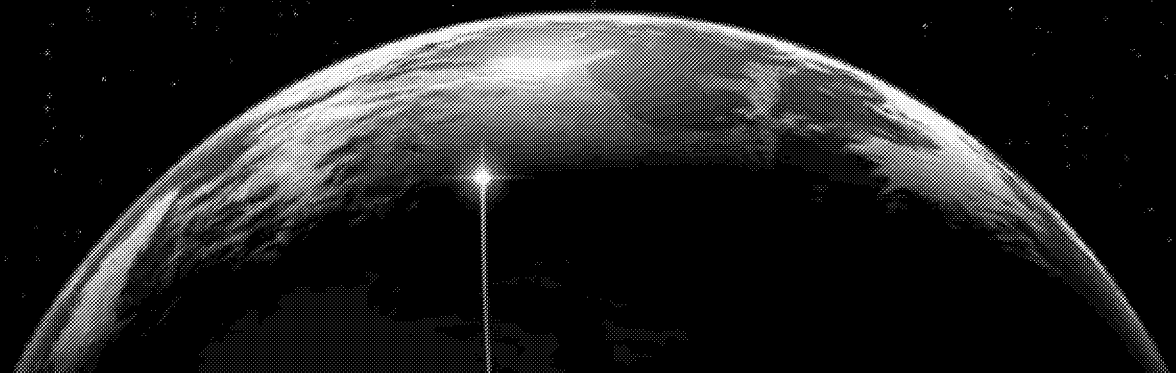
Developing a Software Technology Roadmap to Enable NASA's 21st Century Missions

Marti Szczur, Chief
Information Systems Center - Code 580
NASA Goddard Space Flight Center
Greenbelt, Maryland 20771

Agenda

Goddard Space Flight Center

- GSFC Space Missions of the 21st Century
- Information Technology Challenges
- Components of a GSFC Solution
- Conclusion

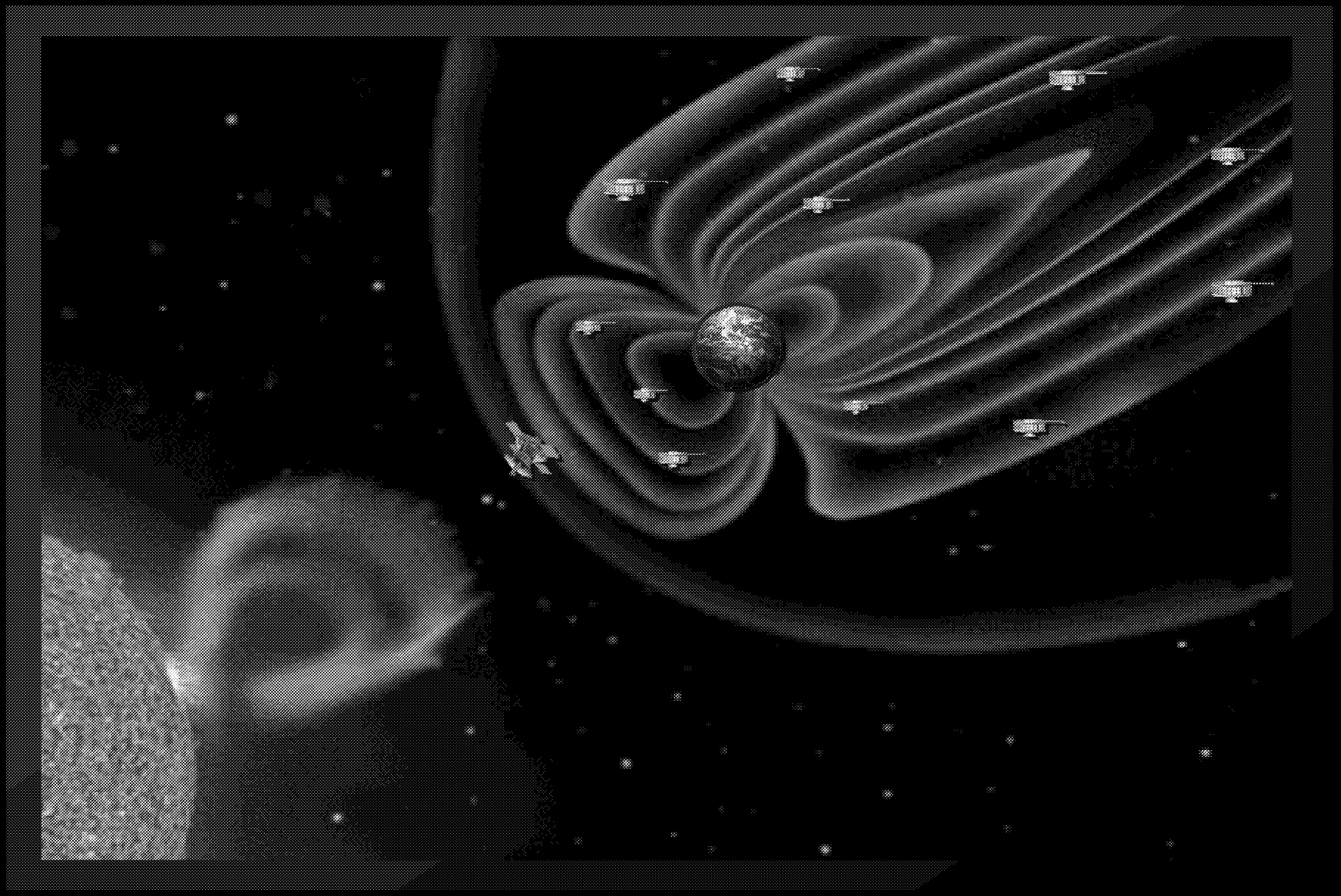


NASA's Technology Program Challenges

Our operating environment has changed significantly...

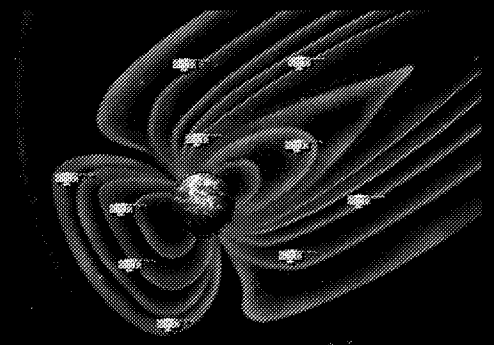
- NASA is more openly "competing" technology funds
- Shift from "technology derived from missions" to "missions enabled by technology"
- NASA has established a goal to significantly decrease mission formulation and implementation time
- Shift focus of in-house work from implementation of current flight missions to development of advanced technologies for future missions
- Funding and Validation Opportunities for mid-TRL technologies
- Increased focus on efficiency, results, and ROI
- "Internet" time scale for technologies and technology development

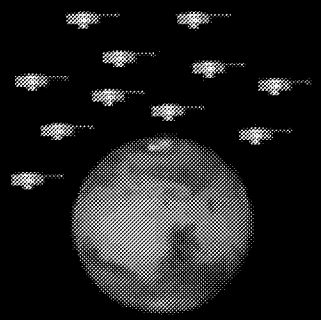
Advanced Space Systems (2000 - 2010)

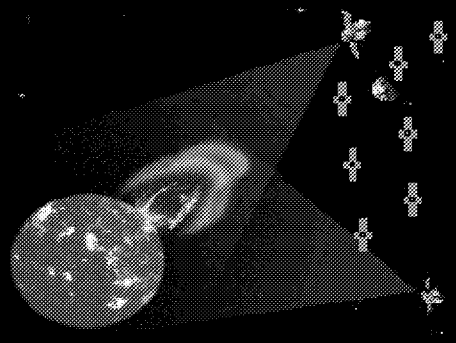


Constellations for Distributed Multipoint Sensing

Space Science Enterprise- Sun Earth Connection Constellation Missions Roadmap







Cost ($M)
1
10
100

Number of Spacecraft
100
10
1

Mass (kg)
10
10
0
1000

Near-term Missions:
- Magnetospheric Multiscale
- Geospace Electrodynamic Connections

Mid-term Missions:
- Magnetospheric Constellation
- Inner Magnetospheric Constellation
- Dayside Boundary Constellation
- Global ITM Constellation

Far-term Missions:
- Solar Flotilla
- Inner Heliospheric Constellation
- Outer Heliospheric Radio Imager
- Planetary Constellations

5 10 15
Years

Revolutionize the multi-point remote and in-situ scientific investigations.

NMP ST5 (Nanosat Constellation Trailblazer) Project Concept

Technologies to be Validated:

Miniature Spacecraft

- Systems Design Integration and Test Technologies

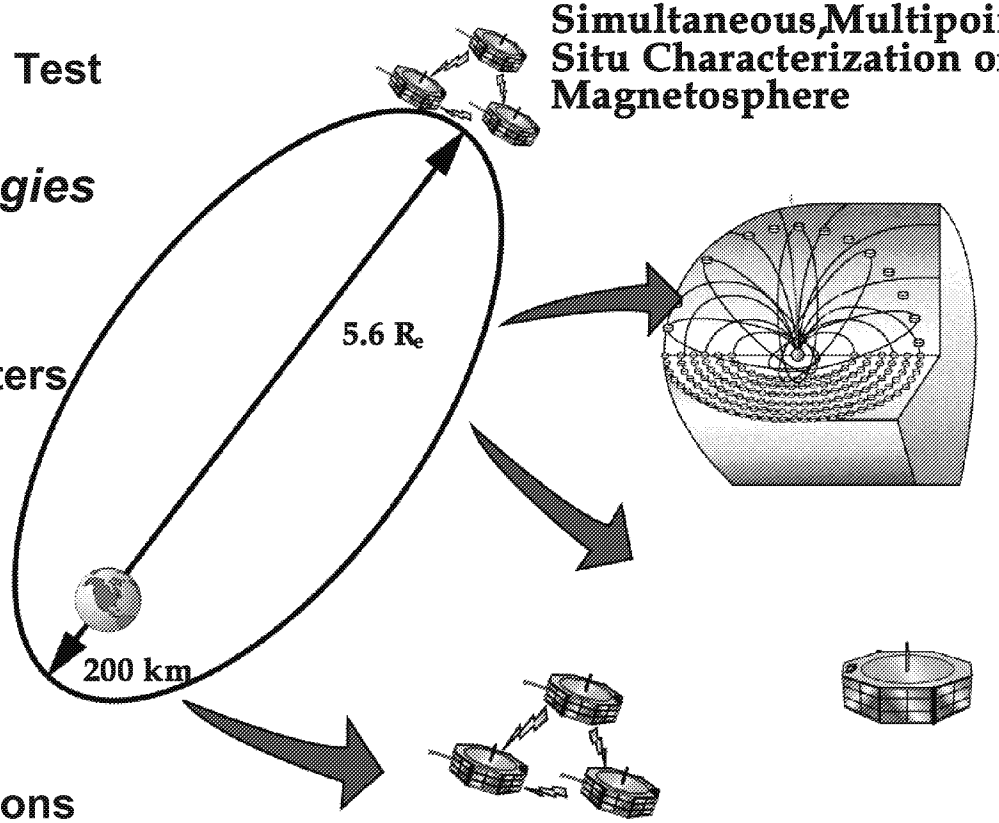
Candidate Spacecraft Technologies

- 5V bus - 1/4V logic
- Li-Ion batteries
- Miniature Transponder & Thrusters
- Multi-functional structure

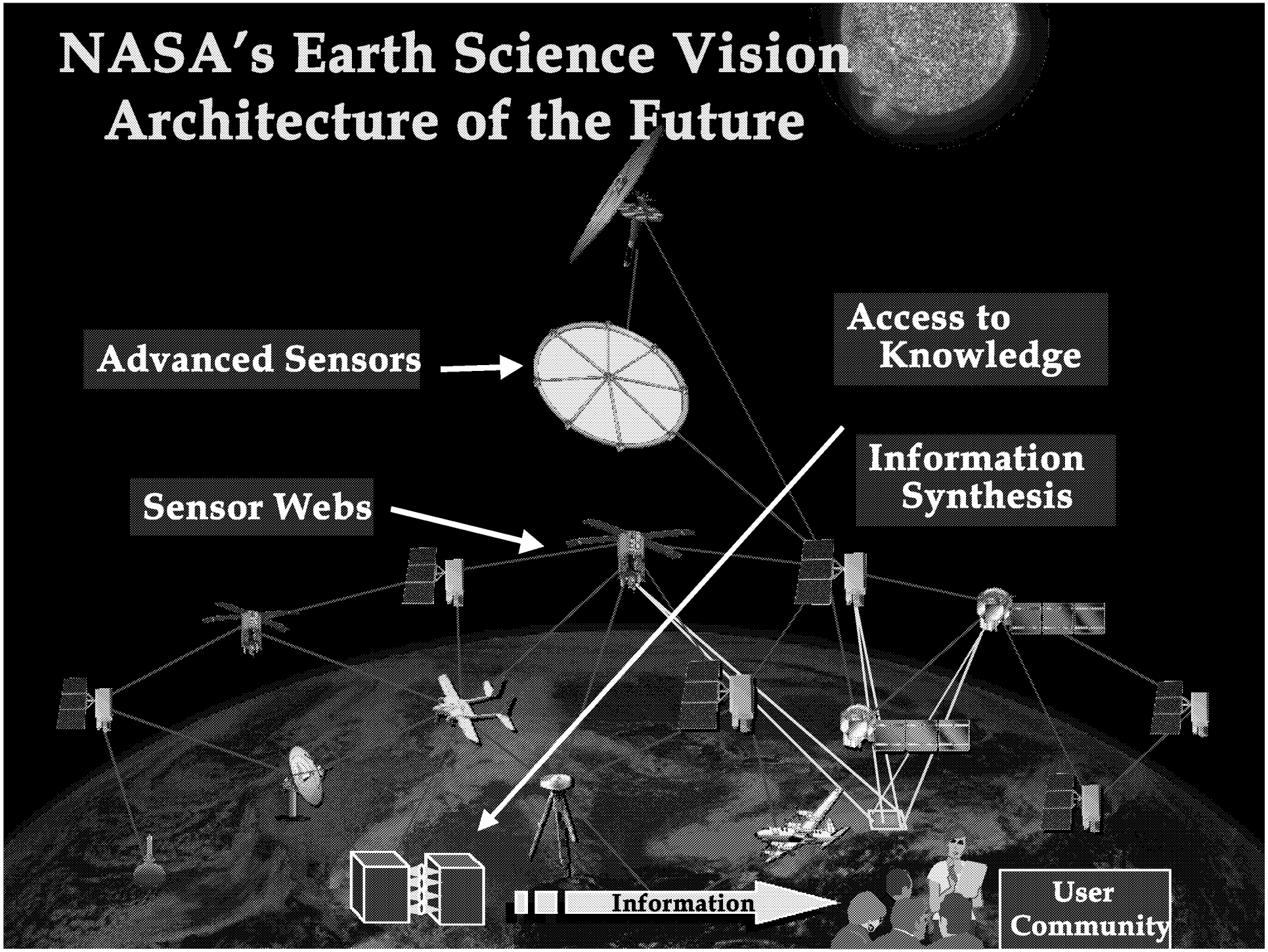
Constellation Control, Coordination, and Operations Architecture

- Ground system autonomy
- Relative ranging
- Intra-constellation communications

Constellation Class Missions
Simultaneous,Multipoint, In-Situ Characterization of the Magnetosphere



NASA's Earth Science Vision
Architecture of the Future



Summary of GSFC Space Missions of the 21st Century

- These missions are characterized by
  - constellations of miniaturized spacecraft
  - advanced sensors & instrumentation
  - innovative communication schemes
  - increased levels of automation
  - onboard processing
  - mission autonomy

- Programmatically, these missions will be noted for dramatically decreased budgets and mission development lifecycles.

- The process by which we conceptualize, research, develop, validate, and infuse technologies must be dramatically improved and accelerated.

To achieve these future challenges, advanced technology– especially IT –will provide a critical role for the formulation, implementation and execution of these missions.

Future Mission Information Technology Needs

- Advanced on-orbit data processing/compression techniques
- Advanced Information Systems architectures
- Autonomy technologies
- Large-scale modeling/simulation/image visualization
- Large-scale data management (archival, retrieval, transportation, transformation)
- Advanced communications
- Next Generation Software engineering
- Advanced Scientific Tools and Systems
- Advanced Human/Computer Interface (HCI) Technologies

Significance of IT to NASA's Future

"When people think of space, they think of rocket plumes and the Space Shuttle, but the future of space is information technology. We must develop a virtual presence, in space, on planets, in aircraft and spacecraft."

- Daniel S. Goldin, NASA Administrator

Components of a GSFC Solution

- Community Building
  - Friends of IS, IT- Federation,
  - Cross-Center Alliance - GSFC, JPL, JSC, Ames
- Maintaining and enhancing IS core competency
- IT Infrastructure Investments
- Strong IT R&D Program and Technology Infusion Plan
  - University collaborations
  - Seed funding for IT research initiatives
  - Term appointments of IT industry/academic leaders
- Create a Mission Software Process
- Computer Science/Information Systems Professional Track and Training Curriculum
- GSFC IS Strategic Plan and Implementation
- IT Roadmaps

GSFC Information Science &Technology (IS&T) Mission

Our mission is to innovate and exploit rapid changes in information science and technology to strengthen GSFC's leadership in NASA's Earth and space science.

GIST Mission Themes

Science has no future without IS&T in today's environment

Move IS&T from a perceived mission cost which must be controlled to a valuable and critical mission resource to be capitalized upon

Unify IS&T community to magnify GSFCs ability to achieve its mission

Provide IS&T leadership, integrated planning and technical expertise to capture and implement GSFC's missions and programs

Create an environment that fosters IS&T "breakthroughs"

Strengthen critical IS&T core competencies to establish competitive advantage

Research, invent, and infuse innovative technology

Facilitate low-cost, robust IS&T systems and services to meet or enhance mission requirements

Leverage IS&T's potential to provide opportunities and sharing of knowledge with the public

ISC Strategic Technology Focus Areas
Objectives and Visions

The ISC Strategic Plan is based on 3 major technology focus areas:

1. Rapid Mission Formulation, Design & Execution:

Enabling revolutionary mission concepts through rapid mission formulation, implementation and execution

Vision: Mission scientists and engineers seamlessly evolve science objectives into mission concepts through virtual model to an operational science system

2. End-to-End System Autonomy:

Enabling effortless science collection through autonomous mission systems

Vision: Mission scientists operate, maintain and reconfigure systems from anywhere in order to optimize an on-board observation and maximize science return

3. Advanced Scientific Analysis Tools & Data Systems:

Enabling science knowledge discovery through seamless and transparent access to information

Vision: Academic and research community has continuous and transparent access to data and information for scientific research

ISC Strategic Technology Capability Roadmap*

(Key Capabilities for future Mission Information Systems)

ISC Technology Focus Area	*Near (2003)*	*Mid (2006)*	*Far (2010)*
Rapid Mission Formulation, Design, and Execution	•Low Cost, Internet-based, Secure Distributed Computing Architectures. •Data interface and distribution framework for Subsystem Design and Analysis Tools •System-level, Model-based design	•Collaborative Design & Operations Environments. •Integrated Simulation and Design Systems	•Immersive Virtual Environments for Collaborative Engineering and Science
End-to-End System Autonomy	•Ground-based, Goal Driven Commanding •Science Feature Identification •Automated Spacecraft Health & Safety •Missions operate as IP nodes on the net	•On-board, Goal Driven Commanding drives Reactive Mission Execution •Constellation Management •Mission operates as mobile-IP nodes on the net	•Multi-mission collaboration for Dynamic Science Agenda •Mission operate as mobile-routering nodes on the net
Advanced Scientific Analysis Tools & Data Systems	•User selectable subscription services for Data Distibution •Feature Extraction and Identification Tools (Data Mining & Knowledge Capture) •Platform Independent Visualization & Analysis ToolsReal-time Updates to Science Models •Interface and Distribution Framework for Science Data & Science Models	•Multi-S/C (including Constellations) Data Processing, Fusion & Analysis •Advanced Information Prospecting and Capture Tools •Collaborative Visualization and Analysis tools •Collaborative (Multi-Source) Updates to Science Models	•Seamless, transparent access to distributed data, info. and knowledge ("Virtual Data Communities") •Knowledge Extraction using Natural Language Interface in an Immersive Environment

* v1.2 - 9/13/99

Conclusions

- Exciting and challenging mission concepts on the horizon for NASA & GSFC
- Significant technology research, development, and application required
- Recognition of importance of IT strategic planning to achieve future science mission objectives
- Significant mission dependency on quality software

Challenge

- Routinely produce “no surprise” software
  - 100% accurate software estimation
  - 100% on schedule and within budget
  - 100% bug-free software

plus

- Significantly reduce software development cycle and cost

plus

- Increase the usability of software systems to maximize the human/computer effectiveness (i.e., human computer interaction)

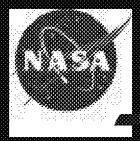
The Impact of Autonomous Systems Technology on JPL Mission Software

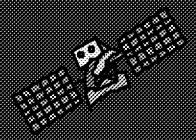
Dr. Richard J. Doyle

Center for Space Mission Information and Software Systems, and
Information Technologies and Software Systems Division
Jet Propulsion Laboratory
California Institute of Technology

24th Annual Software Engineering Workshop
Software Engineering Laboratory
NASA Goddard Space Flight Center

December 1, 1999



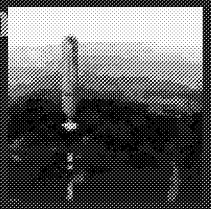


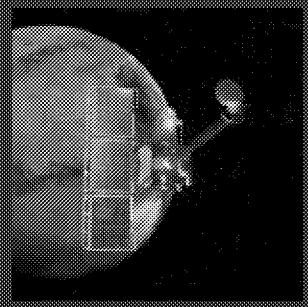
Autonomy for Future Missions

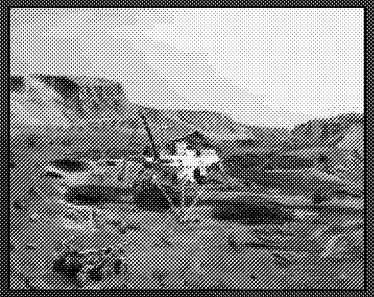
JPL

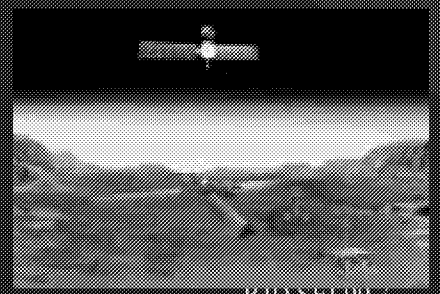
Mars Outposts

- **Remote Science Laboratories**
  - Tele-operated or autonomous laboratories in the planetary environment for handling and conducting in situ scientific investigations on collected samples
  - **Three scales / resolution**
  - remote sensing
  - distributed sensing
  - point sensing
- **Heterogeneous, cooperating networks**
  - distributed networks of sensors, rovers, orbiters, permanent science stations, probes: all of which respond to sensing events, discoveries, changing PI directions, etc., to provide rich presence in Mars environment for science community and public
  - **Infrastructure**
  - Planetary permanent infrastructure to support series of science and/or commercial missions leading to human presence

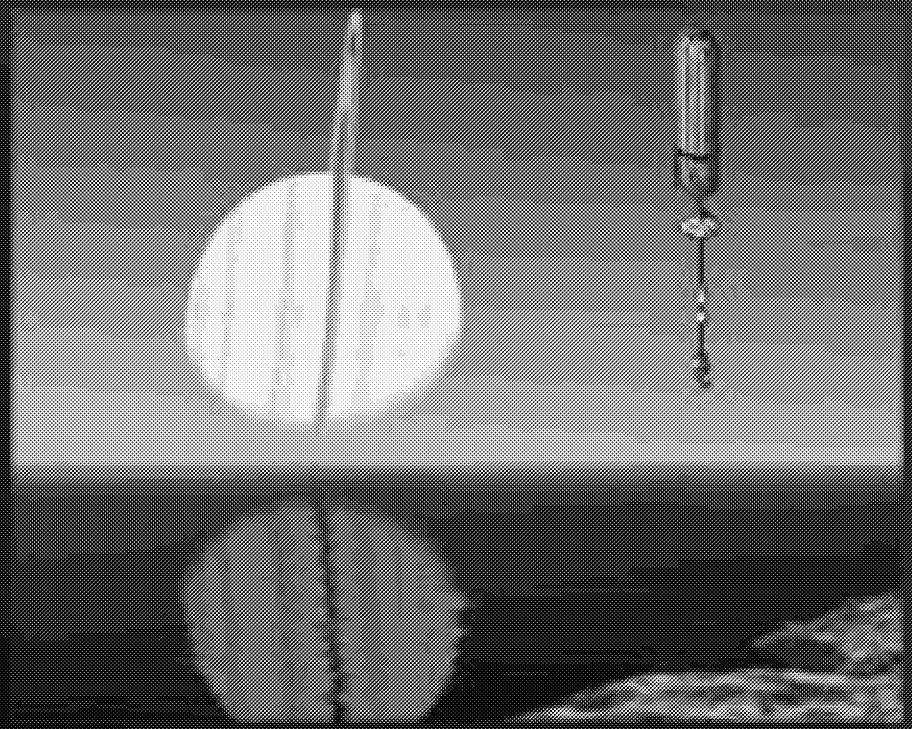








Titan Aerobot



- The aerobot conducts in-situ science operations when landed, and wide-area imaging when aloft.
- Archived and learned models of wind patterns assist path planning, enabling near-returns to areas of high scientific interest.

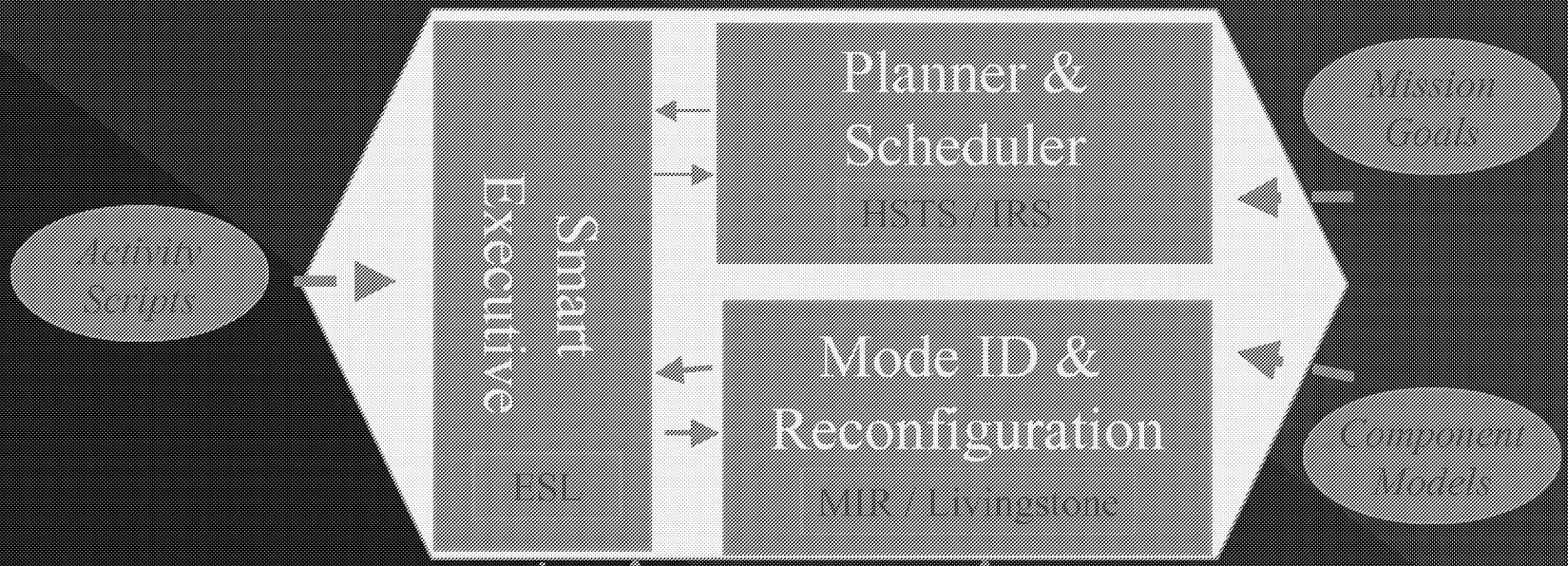
Europa Cryobot / Hydrobot

- Perhaps more than any other, a mission of discovery in a truly alien environment: How to know what to look for? How to recognize it?

The Emergence of Autonomy

JPL

Remote Agent Architecture



Closing Loops Onboard

Ground assistance invoked with focused report on spacecraft context and history

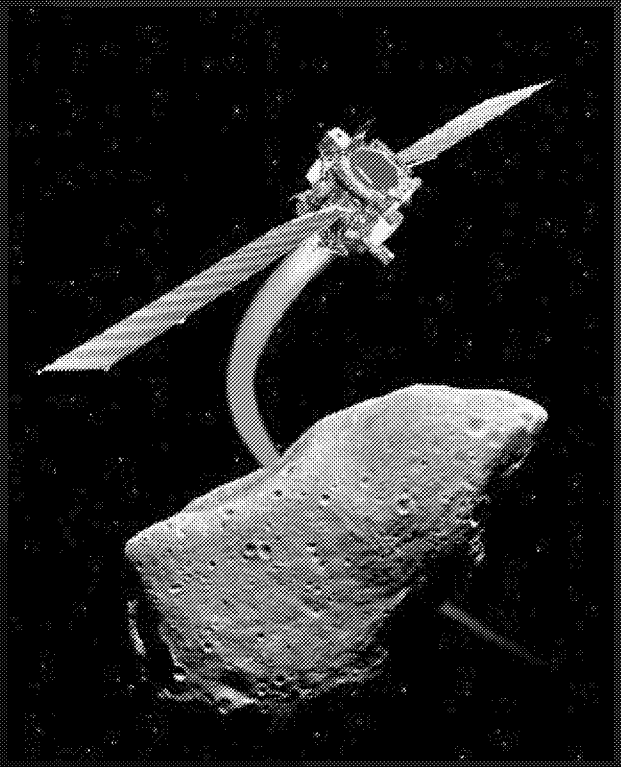
Replanning of mission activities around altered resources or functions

Diagnosis of faults and informed selection of recovery actions

Local retries or alternate, pre-defined activities to achieve same goal

Several layers of onboard recovery provides for unprecedented robustness in achieving mission goals in the face of uncertainty

New Millennium Flight Experiment



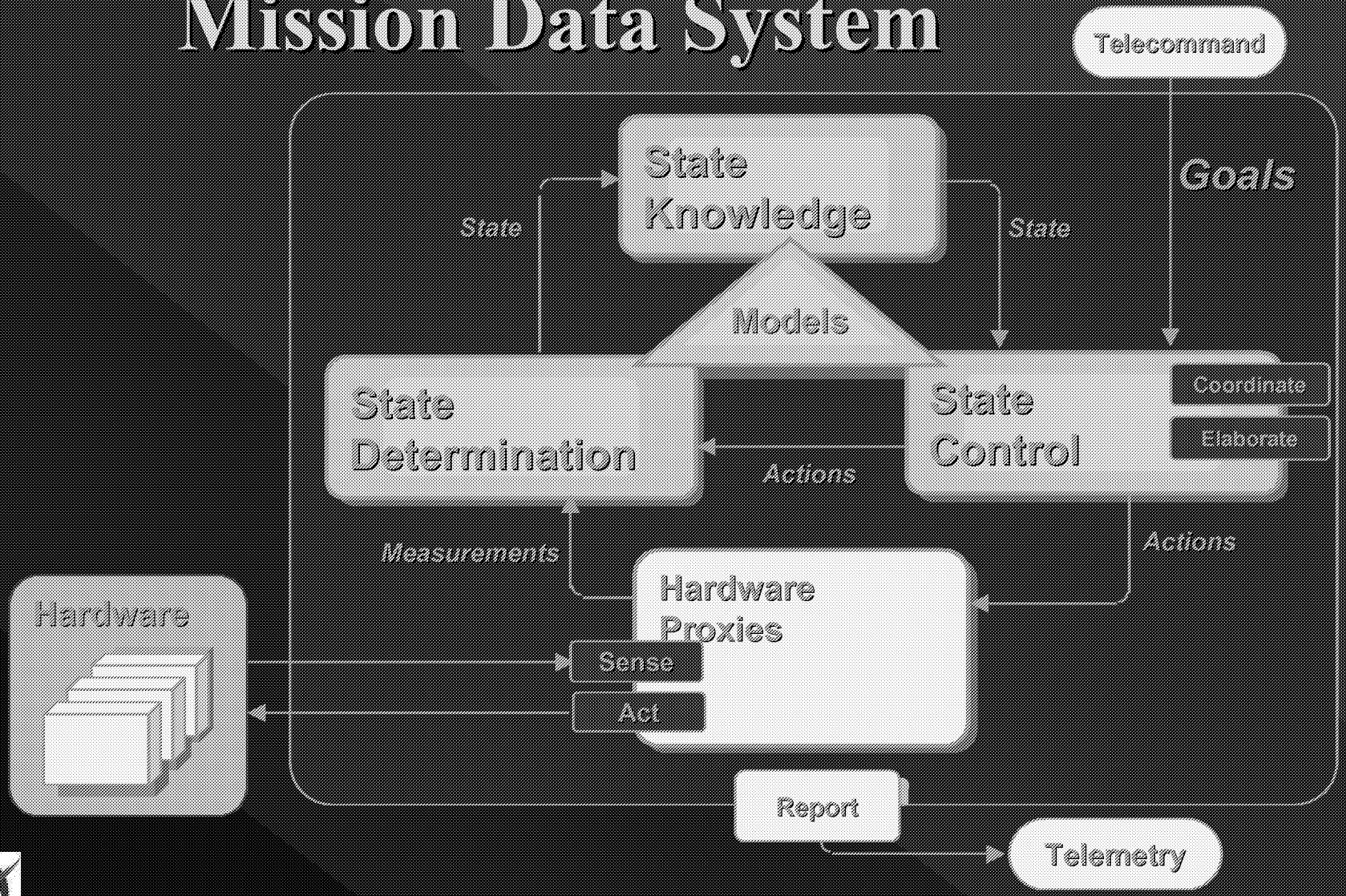
- **DS-1 has encountered an asteroid and will encounter a comet.**
- **Remote Agent Experiment (RAX) achieved 100% of its technology demonstration goals in May '99.**
- **RAX joined eleven other DS-1 technology experiments such as onboard optical navigation and solar electric propulsion.**

Software Engineering Challenges

SEL

R. Rasmussen et al

Influence of Remote Agent: Mission Data System

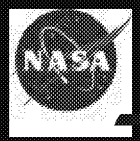


MDS Architectural Themes

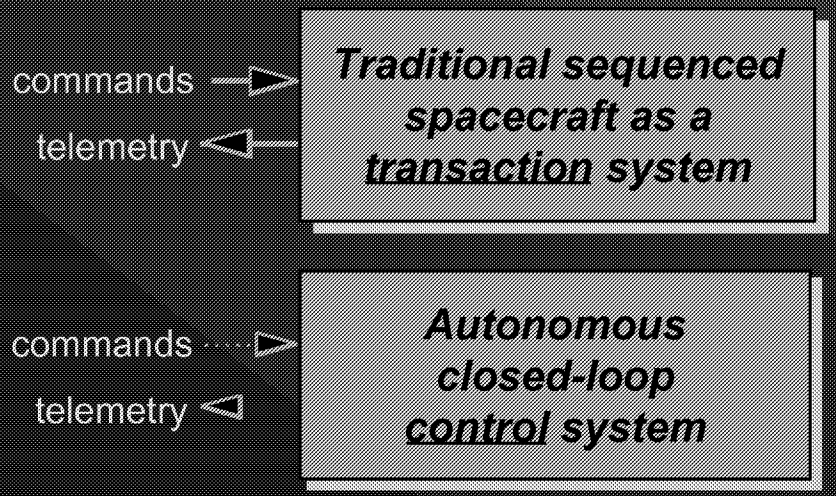
- Unifying state-based paradigm behind all elements
- Extensive and explicit use of models
- Goal-directed operations specifies intent, simplifies workload
- Closed-loop control enables opportunistic science gathering
- Fault protection is natural part of robust control, not an add-on
- Explicit resource management (power, propellant, memory, etc)
- Navigation and attitude control build from common base
- Clean separation of state determination from control
- State uncertainty is acknowledged & used in decision-making
- Clean separation of data management from data transport
- Upward compatibility through careful design of interfaces
- Object-oriented components, frameworks, design patterns

Scalable Autonomy

Capability	*Baseline* ⟷	*Greater Autonomy*
Planning & Scheduling	Plans generated and validated on ground; some automation	Plans generated onboard from uplinked goals within s/c and environment context
Execution	Highly predictable sequences fully compiled to a timeline	Flexible, deferred commanding; multi-threaded execution; local recovery and cleanup
Fault Protection	Fault identification puts spacecraft in safe hold; mission suspended	Model-based diagnosis and recovery for overall s/c state; mission continuation enabled



Autonomy Software Validation



Key idea: (borrowed from model-based fault diagnosis)

- Do not attempt to enumerate all possible software failures
- Rather, define and identify departures from acceptable bounds on software behavior
- Apply at design, test and run time

Analytic Verification Technology

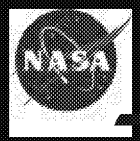
(Automated Software Engineering Group, NASA Ames)

- Highly autonomous systems typically perform numerous *concurrent* activities, e.g., science observations, instrument calibration, fault monitoring & diagnosis, activity planning, etc.
- Mission systems built upon MDS will employ multi-threaded execution, with an *enormous* space of possible states and paths through those states.
- Concurrent interacting programs are particularly vulnerable to *synchronization bugs* such as race conditions and deadlocks.
- ARC is applying and developing analytic verification technology (a.k.a. "model checking") to mathematically analyze specifications, code, and models for consistency with requirements and designs:
- At JPL, for Mission Data System (D. Dvorak)
  - At GSFC, for the Advanced Architectures & Agents Group (J. Breed)

ARC

Autonomy and Software Engineering

- New critical-path challenges for software engineering are entailed by the autonomy capabilities required for many future NASA missions:
- Verification and validation
  - Reliability
  - Flight / ground architectures
  - Technologies such as auto-code generation
- The specific drivers emerging from the autonomy area are part of a general pattern of increased importance of software engineering to achieve quality mission software



Session 5: Using the Experience Factory

Frank McGarry, Computer Sciences Corporation

Ross Jeffery, University of New South Wales

Carolyn Seaman, University of Maryland

Attaining Level 5 in CMM Process Maturity

Frank McGarry
Bill Decker
Joe Haskell
Amy Parra

Abstract

In November 1998 the CSC SEAS Center achieved the rating of CMM Level 5 and became the sixth organization in the world to have ever attained that goal. The Capability Maturity Model (CMM) (Reference 1) is a worldwide recognized benchmark of process maturity for software organizations and is used to assess the quality of an organization's software process. During the period covered by this study, the SEAS Center comprised approximately 850 personnel supporting systems engineering, software development, and analysis for NASA/GSFC. During the years of continually improving the processes toward the goal of attaining the level 5 rating, detailed information was recorded, tracked and analyzed so that subsequent efforts by other CSC organizations could benefit from the experiences of SEAS. This paper is a direct result of the collection and analysis of that process experience data.

This paper begins with a brief overview of the SEAS organization that emphasizes the aggressive process improvement approach that has been in place since 1994. The paper will discuss the coordination of improvement initiatives, the role of goals and industry benchmarks, the organizational strategy and the use of key documents in measuring improvements. Additionally, the investment and benefits of an improvement program are discussed. Finally, based on the SEAS experience, the paper presents seven key factors that are the recommendations for any software organization undertaking an aggressive process improvement program.

Section 1 Background

CSC is a major software integration and services provider with over 50,000 employees in offices worldwide. The Systems, Engineering, and Analysis Support (SEAS) Center is part of the Federal Sector and comprises approximately 850 persons supporting the National Aeronautics and Space Administration (NASA) at the Goddard Space Flight Center (GSFC) in the disciplines of systems engineering, development, maintenance, and analysis (Figure 1-1).

CSC has supported NASA in the GSFC environment since the 1970's. Staffing at the Center has varied from 700 to 1700 over the last 10 years. The SEAS Center is organized as a program with central offices supporting program management (PMO), process engineering (PEO), quality assurance (QAO), and program control (PCO). Software configuration management is typically a project responsibility and subcontracting for product development is very rare. The number of projects within the program varies but is typically about 20. Approximately 50% of the organization is directly involved in the software development or maintenance activity.

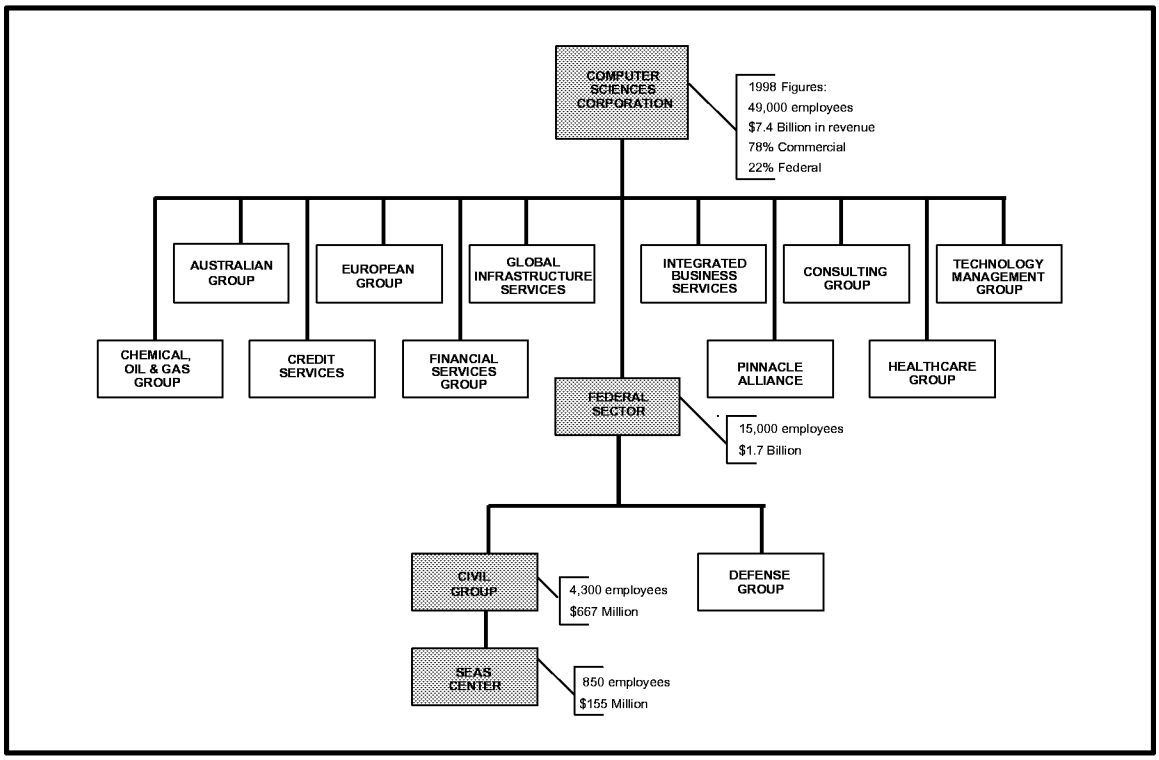


Figure 1-1 SEAS Center Within CSC

Because of the growing importance of establishing process maturity within software intensive organizations, the SEAS Center initiated an aggressive process improvement program in 1995. A process improvement plan with specific goals was written to guide the initiative. Of the goals, four were product goals with objective measures

(productivity, quality, predictability, cycle time), and another goal specified compliance with standard industry benchmarks.

The processes used to support the work on SEAS have always been regarded (by the CSC staff) as being good processes although an early external evaluation of the processes produced a Level 1 CMM rating in 1991. Despite this early discouraging result, the Center continues to view benchmark evaluations as an important activity supporting process improvement efforts (Figure 1-2).

After some success with internal process audits and CMM self-assessments, SEAS Center adopted the use of evaluations against industry benchmarks conducted by independent consultants. The 1995 process improvement plan included goals for both CMM and ISO 9001 (hereafter referred to as ISO) evaluations.

The results of benchmarking activities are summarized in Table 1-1. In 1998, the SEAS Center became the sixth organization in the world to be rated at CMM Level 5 and the first organization to be both CMM Level 5 and ISO registered.

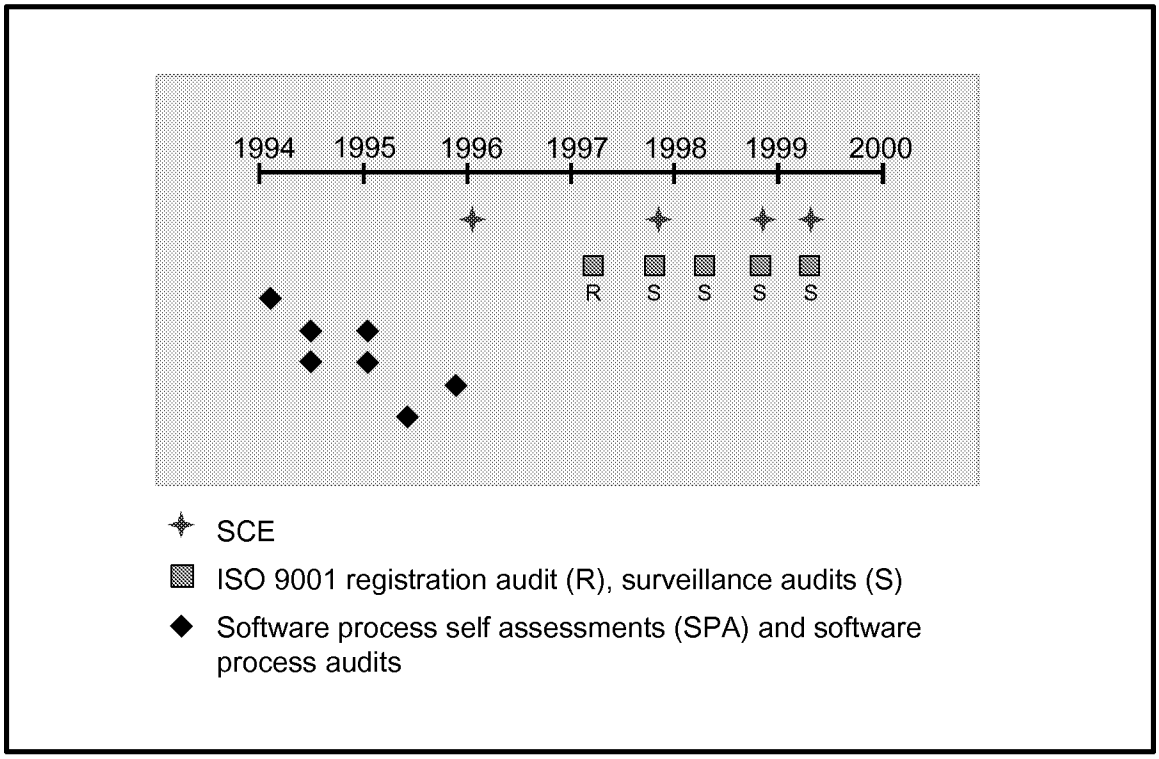


Figure 1-2 SEAS Center Benchmarking History

	SCE (2/96)	ISO (5/97)	SCE (11/97-1/98)	ISO (11/97)	ISO* (6/98)	SCE (11/98)	Part of 'Group' SCE (5/99)
Preparation time	• 4 months	• 12 months	• 2 months	• 6 months	• 1 month	• 4 months	• 1 month
Organization effort	• 1800 staff hours	• 3400 staff hours	• 800 staff hours	• 500 staff hours	• 200 staff hours	• 800 staff hours	• 60 staff hours
Use of external consultants and training	• None	• 200 hours - Consultant - Internal auditor training - Pre-registration assessment	• None	• None	• None	• 2 staff days	• None
Preparation strategy	• Perform gap analysis • Use lessons learned from 1991 SCE • Focus on deployment	• Develop implementation plan • Use external experts • Train staff • Focus on deployment	• Complete actions items • Provide awareness seminars • Use internal assessments	• Continue process improvement initiatives • Focus on management review, internal audits, and corrective actions	• 1 refresher lecture • Internal audits continued	• Study/learn specific details of level 4-5 • Advance collection of evidence • Group seminars • Mock SCEs	• Update 'evidence' archives • Group seminars
Results	• 13 of 18 KPAs satisfied	• ISO registration achieved	• CMM Level 3 achieved Levels 4-5 (2 of 5 KPAs satisfied)	• ISO registration maintained (2 minor findings)	• ISO registration maintained (3 minor findings)	• CMM Level 5 achieved (18 of 18 KPAs)	• All reviewed KPAs satisfied

* Additional ISO assessments held in 11/98, 5/98, and 11/98

Table 1-1 Summary of Benchmarking Activities

Section 2 Approach

As discussed in Section 1, SEAS had an extensive legacy of process development and improvement at the time that it achieved CMM Level 5 in 1998. SEAS process development work during the late 1980s and early 1990s consisted primarily of refinements of the SEAS System Development Methodology (SSDM) and its supporting standards and procedures (S&Ps). Such refinements were recommended by process users and approved by senior management. This bottom-up approach worked reasonably well and resulted in the establishment, deployment and use of SSDM and approximately 100 S&Ps.

Between 1989 and 1994, 508 proposed changes to SSDM were submitted by process users; of these, 379 were implemented in whole or in part. Unfortunately, most concerned relatively minor adjustments to existing processes. SEAS management noted three major flaws in this process improvement strategy: (1) a formal "learning through experimentation" process was not being used, (2) establishment and measurement of goal achievement was weak, and (3) SSDM and its associated S&Ps were becoming obsolete since new approaches and methods were not being adequately integrated. A new approach was needed.

During the early 1990's the Quality Improvement Paradigm (QIP) (Reference 2) was being used in the SEAS Software Engineering Laboratory (SEL). (The SEL, Reference 3, is a joint venture involving CSC, NASA and the University of Maryland.) The QIP, shown in Figure 2-1, established a framework for improving SEAS by treating projects as experiments, packaging results, and making such results available to all SEAS projects. The QIP eliminated the process improvement flaws noted above and was accepted by SEAS management as a solid foundation upon which to build the SEAS improvement program. Since 1994 the QIP has served as the model for process improvement for the SEAS Center.

The SEAS adoption of the QIP as its improvement model focused attention on improving key activities such as communication, coordination, establishment of goals, measurement of change, and experience sharing. Thus, attention was redirected from refinement of existing processes to making SEAS a learning organization based on the experiences of its projects. Adoption of the QIP radically changed how improvement was addressed by the organization. Some of these changes are briefly discussed below.

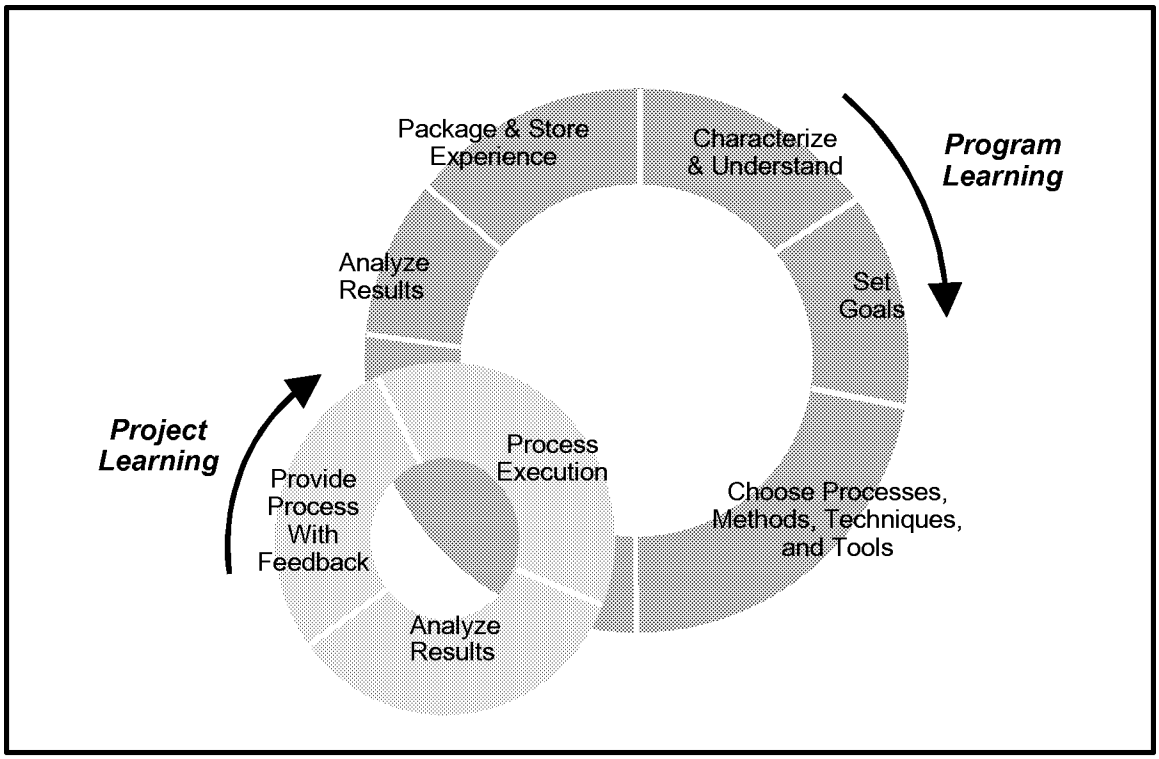


Figure 2-1 Quality Improvement Paradigm (QIP)

1. SEAS-Level Coordination of Projects' Process Improvement Initiatives.

The QIP is based upon the assumption that a Program-level group is aware of project-level experiments, provides guidance to projects, and makes successes and failures known to other projects within the Program. For SEAS, responsibility for this type of coordination was assigned to the PEO. Use of "shepherds" and weekly 'Process Deployment Team Meetings' as described below directly resulted from adoption of the QIP.

- Shepherds are typically Process Engineers or Quality Assurance personnel who are aware of activities and project experiments throughout the organization. The shepherds are assigned to work directly with a project to guide process implementation and avoidance of problems experienced by prior projects. The shepherds perform as project support personnel in responding to needs of the projects in tailoring, understanding, and implementing processes appropriate for the project.
- Process Deployment Team Meetings are weekly 1-hour meetings held to discuss some aspect of the SEAS processes. The meetings are facilitated by a process engineer and attended by all levels of management and some personnel from the projects. Typically, these meetings take the form of a briefing followed by questions, answers and comments regarding the given topic. Topics have included; top 10 steps in adopting mature processes, effective use of measurement, how ISO and CMM are related, impacts of inspection techniques for software, how to set goals in project planning, effective risk management, how our processes conform to Level 5, and

results of recent project experiments presented by project personnel. The meetings are interactive, with all participants joining in the discussion.

2. Establishment of Product-related Goals Rather than a Goal of Compliance with Industry Benchmarks

The QIP requires establishment of goals. For organizations such as SEAS, compliance with industry benchmarks is an important business goal. Much of SEAS process-related work in the early 1990s was directed to the goal of demonstrating compliance with the CMM. However, once the QIP had been adopted as the improvement model, SEAS goals evolved from a focus on complying with industry benchmarks to a focus on improving products and achieving customer satisfaction. Project buy-in to use of the QIP was easily achieved once projects appreciated the value of learning to improve their products based on the experiences of prior projects.

3. Use of Industry Benchmarks as Tools to Achieve Product-Related Goals

SEAS established ISO-9001 as its primary tool for guiding and measuring improvement. Similarly, the SEI CMM served as a tool for measuring progress in improving the SEAS software development processes. ISO requires participation by all elements of the organization, in contrast to the software development focus of the CMM. However, ISO-9001 and the CMM are complementary and support the product improvement strategy as embodied in the QIP. (As a byproduct, use of ISO and CMM support senior management's business goal of compliance with key industry benchmarks.) Industry benchmarks such as ISO and the CMM served as gates for verifying process maturity and use. Use of external assessors ensured objectivity in measuring progress toward achievement of goals related to compliance with industry benchmarks.

4. Use of 'Separation of Concerns' Strategy

Project personnel were not required to become familiar with the details of the QIP or industry benchmarks; deployment of the QIP, ISO, CMM and other strategies was assigned to the process engineers. This left projects free to focus their limited resources on improving their products and services rather than on complying with industry benchmarks. As discussed above, the shepherds provided guidance to projects in applying the QIP and complying with the industry benchmarks.

5. Document Organization Profile and Improvement Goals

Application of the QIP requires an understanding of current product characteristics (defect rates, cycle time, accuracy of estimates, etc.) and improvement goals. Therefore, consistent with the QIP, SEAS documented its organizational and product characteristics in a profile document (Reference 4) and established SEAS-level improvement goals in a process improvement plan (Reference 5). These documented "where we are" and "where we want to go", and served as the roadmap for measurable process improvement. The SEAS Quality Management System Manual (Reference 6) documented the roles and responsibilities of each SEAS group in achieving improvement.

The reader should be cautioned that the QIP worked well for SEAS and would likely work well for other organizations. However, for maximum effectiveness, it should be applied with consideration given to the culture and maturity of the organization. For SEAS the approach was to focus on identification and deployment of a formal model since basic processes were already in place. Recommendations for other organizations are provided in Section 4.

Section 3 Return on Investment

In order to determine the value of investment made toward process improvement, the SEAS Center measured impacts of improvements in three areas: 1) impacts to the performance of the organization 2) impacts to business opportunities and 3) impacts to the products generated. This set of measures of 'return on investment' was used to continually mold the program of process improvement and to help determine which areas of improvement should be the focus for continued efforts. They were also used to make a determination as to whether or not the process improvement program was worth the investment of time and resources and whether or not the program should be continued or modified. The value of the process program was measured against the cost of the overall program. This value of the program compared to the investment cost is what we term 'return on investment'.

3.1 Cost of Process Improvement

The cost of the process program was tracked by maintaining detailed records of the effort expended by staff carrying out activities directly on the program (Process Engineering staff as well as Quality Assurance staff) and also including indirect effort required by the project organization in attending special training sessions or attending special audit activities. The tracked costs include developing processes, deploying, measuring, training, maintaining (packaging), developing infrastructure, and process improvement. The costs do not include project operations performing CM, QA, planning, etc., but do include their cost of participating in studies, training, audit participation.

For the period July 1994 through November 1998 (the date when the Level 5 was attained) the cost of the process improvement program was approximately 30 staff years of effort. This cost was primarily the cost of the organization's process engineers responsible for defining and carrying out the improvement program. Fairly detailed records were kept in order to track this expenditure. Records of costs permitted the analysis of the distribution of effort across different functions and the shift of allocation from early months of the program to later months of the program.

The records of costs categorized the effort by 5 main areas of activity: (1) writing and maintaining written processes, (2) deployment of processes (working with projects via training and direct help in using processes), (3) creating and maintaining the infrastructure of processes (data bases, libraries, etc.), (4) planning improvement including the writing of plans, carrying out studies and analyzing measurement, and (5) reporting and participating in reviews of the process program.

Table 3-1 shows the distribution of the effort for these 5 major activities. Overall, the highest percentage of effort was allocated to the deployment activities. Process engineers focused on getting the defined processes into practice (shepherding) as opposed to only focusing on generating and maintaining the written standards, processes, methodology, etc.

Activity	4 year cost	1995-1996	1997-1998
Develop/Maintain Processes (write/update)	6 SY	40%	15%
Deploy/Training/Awareness	10 SY	10%	40%
Infrastructure (data base, libraries, distribution)	2 SY	5%	5%
Process Improvement (planning, studies, experiments, analyzing)	8 SY	15%	30%
Assessment Preparation (SCE, ISO)	3 SY	25%	5% - 5%
Reporting/Reviews	1 SY	3%	3%

Table 3.1 Cost Distribution for Process (For Organization of 800 Persons Over 4 Years)

Table 3-1 also indicates a shift in emphasis from the writing and refining written processes to the emphasis on deployment of process. The shift reflects that over time the process engineers realized that the largest value of the program was in interacting directly with the projects and not in merely producing and enhancing written processes.

3.2 Value of the Process Improvement

As mentioned in the introduction, the impact of the process program was measured in three areas: value to the organization, value to business opportunities, value to the products generated.

3.2.1 Impact to the performance of the organization

The first measure of the impacts of the improvement program was a determination of perceptions, general performance and structure of the organization as a whole. In general, it is a determination as to whether or not the personnel viewed the program and the changes as a value to their own projects performance. This was determined by taking surveys, interviewing project personnel and managers and by soliciting feedback from customers.

There were significant favorable impacts to the overall enterprise characteristics of the SEAS organization. These changes included both technology enhancements as well as operational impacts that supported a more efficient and effective structure. Specific

impacts that were identified by both project personnel as well as managers across the organization included:

1. The process improvement program resulted in a focus of achieving common goals for SEAS. With the formal improvement plan generated and with specific goals identified as part of the plan, there was a foundation established for all SEAS personnel to contribute to improvement. The improvement goals and overall program prompted project personnel to contribute to the overall SEAS improvement program as opposed to only their own project program. This was supported through the management reviews, process meetings, progress reporting and assessments (both internal and external) that were included as part of the program. The improvement program promoted the concept of SEAS operating as a well disciplined enterprise rather than a set of individual projects with local goals and challenges only.

   This fact of operating as an integrated organization also improved the communication between projects (sharing lessons, improvement ideas, measurement approaches, and tailoring approaches for SEAS processes).

2. The improvement program added a strong discipline for all projects to adopt and adhere to SEAS processes. The improvement program included the use of formal assessments such as ISO audits, SCEs, and internal audits. With the use of regular formal assessments and with the strong senior management support of the improvement program, all projects within SEAS had strong incentives to adhere to the processes and disciplines defined by SEAS.

3. The improvement program resulted in a significant upgrade and improvement to the set of SEAS standards, policies, and processes. Since the program adhered to the concept that changed processes should be driven by needs and experiences of projects (as opposed to being changed to meet an external benchmark) and since ISO stressed the value of producing processes that were short, crisp and directed to the actual needs of the projects, the set of SEAS standards and processes were revised with a focus on project need and SEAS lessons learned. This resulted in a set of processes that the projects felt were much more in keeping with their specific needs.

4. The improvement program promoted an accelerated adoption of needed technology change. The activities of the improvement program included the continual search and incorporation of enhancements that would lead to more efficient development and operations. There were several technology changes that were driven by this approach to sustain change. Such enhancements as the universal adoption of on-line, electronic documentation and the adoption of common CM tools were prompted by the improvement program. The goal of attaining full ISO registration was more easily addressed by producing complete on-line, electronic documentation.

5. The accomplishments resulting from the improvement program produced a sense of pride and accomplishment for the entire SEAS organization. The recognition that SEAS received by achieving ISO registration and by attaining high maturity ratings with CMM was shared by all SEAS personnel. Since all projects and personnel participated at some level, the entire organization felt the recognition received was something that each of the individuals could be proud.

3.2.2 Impact to business opportunities

The second measure of value of the process improvement program is the impacts it had on business opportunities. The improvements demonstrated by the SEAS program played a major role in winning new business for CSC. The improvement program in general demonstrated to potential clients that CSC was very serious and committed to process improvement. This fact alone can be a discriminator in selecting a support contractor. It is important that clients see a demonstrated program of sustained improvement.

In addition to demonstrating an aggressive improvement program, CSC could point to the levels of achievement recognized by CMM and by ISO. These achievements are frequently used by potential clients in scoring capabilities of contractors. In the case of the SEAS achievements, at least 3 programs used the independent ratings (ISO and CMM) and the established processes as consideration in selecting CSC for additional work. The additional work in 1999-2000 amounted to over $500M in contract value. The established SEAS processes were identified as key elements of the new work.

3.2.3 Impact to the software products

Probably the most important measure of success of any improvement program is the measure of product improvement. Have the products and services been favorably impacted by the changes made to process?

The SEAS improvement plan identified 4 product measures that were part of the goals of improvement. The product measures included productivity, defect rates, cycle time, and estimation accuracy. From the start of the program in the Summer of 1994, detailed measures, records, and general information were recorded for the purpose of guiding the change and for tracking impact of any changes that were made. Details of the measures that were tracked and the results of analyzing the changes to the product measures were reported in 1998 at the time of the Level 5 rating. Details of these results can be found in Reference 7.

By reviewing the detailed process ratings over many years along with the detailed product data (productivity, defect rates, etc.) an attempt was made to statistically determine the correlation between the process changes (increasing maturity level) and the product changes. The analysis showed a constant 6 percent/year improvement for both productivity and quality from the start of the program through the end of 1998. (See Figure 3-1). Further analysis showed that there was also a 6 percent/year improvement from 1987 through 1994. Attributing sustained product improvement to process change from this evidence is not conclusive. Improvements in technology, personnel, environments, as well as process change, remain as possible sources of the observed product improvements.

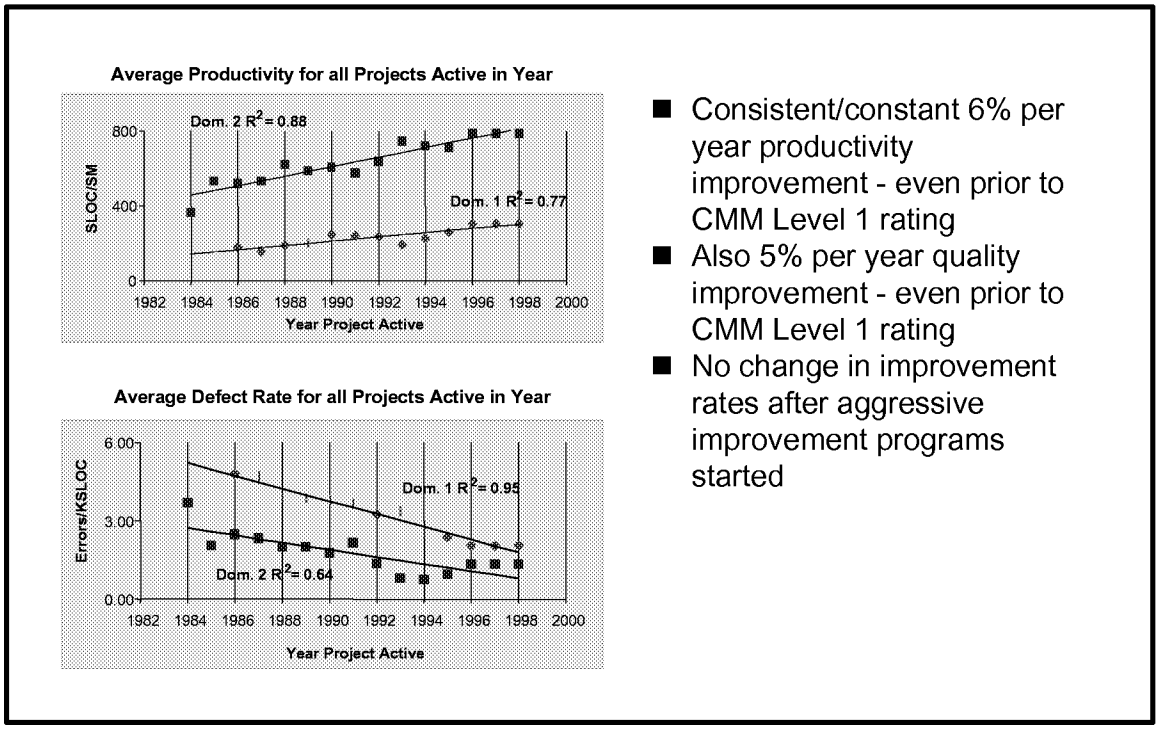


Figure 3-1 Productivity and Quality Trends Over Time

When an attempt was made to correlate process maturity of projects with the product measures (Figure 3-2) there was no statistically significant result. The correlation was computed from data extracted from SCE reports generated for each project. Each project was reported compliant, partially compliant or not compliant with each Level 2 and 3 Key Process Area (KPA). From this data, each project was assigned a maturity 'score' on a scale from 1 to 3. Product measures and the maturity 'score' were analyzed for a correlation between high maturity ratings and the high performance of each of the product measures (quality, productivity, cycle time and predictability). Correlalations were all of low significance; the R^2 values ranged from a low of 0.15 to a high of 0.49. There is not a clear explanation for this, but the authors surmise that the strongest explanation is that process is simply a very difficult parameter to measure in isolation. Using a project's maturity rating as the only measure of process may be too simplistic. Details of this process are explained in much more detail in Reference 7. Work on this analysis is continuing.

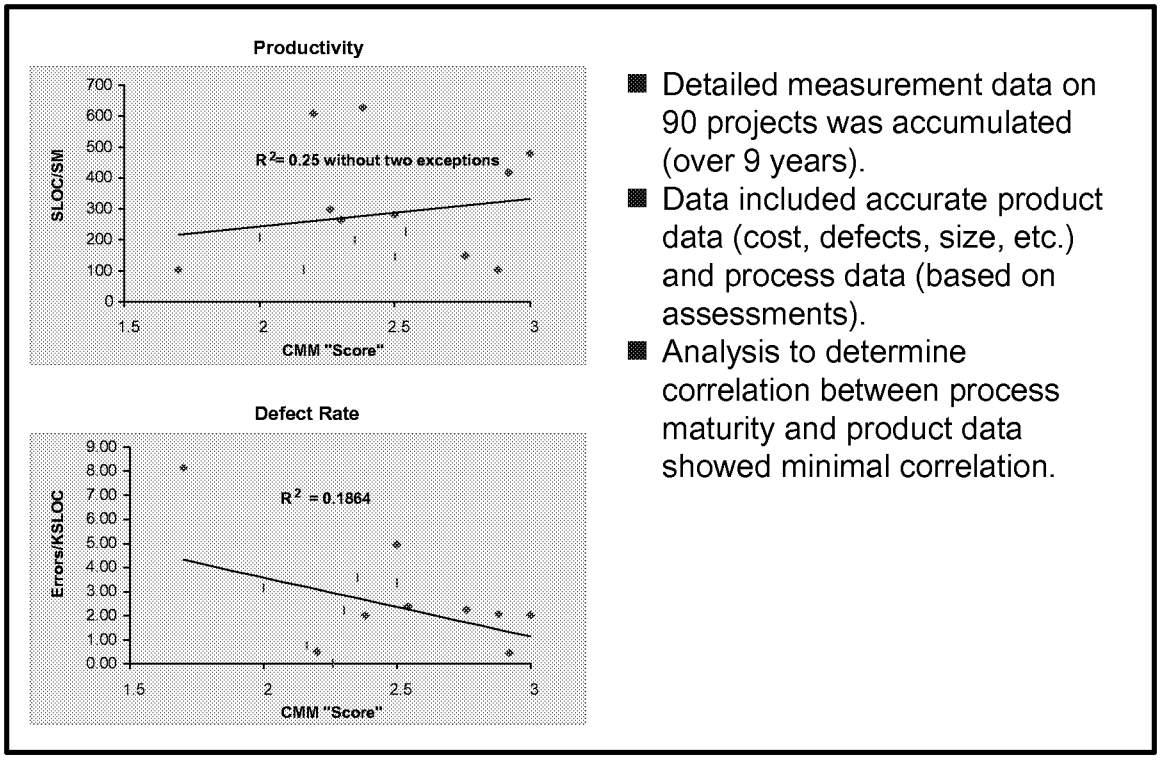


Figure 3.2 Impact of CMM Maturity on Cost, Quality, Manageability

3.3 Relative Impact of Improvement Activities

There were many activities undertaken and many avenues pursued with the goal of attaining the high maturity ratings and demonstrating improvements to the SEAS organization. Shortly after the Level 5 rating was achieved, a review of the lessons, activities and steps was held in an attempt to determine which of the steps seemed to be of most significant value (and which seemed to be of minimal value).

Sources of information included surveys collected from project developers and managers, lessons learned reports generated periodically during the 4-year initiative, interactive workshops held (as part of the regular 'Process Deployment Team Meetings'), and interactive discussions held with the process and quality assurance personnel. Personnel were asked to identify which activities had the most favorable impact on improving processes within SEAS as a whole and on projects specifically. Four activities consistently were rated as being the most effective in leading to the success of the process improvement:

- Shepherding
- Process deployment team meetings
- Library building with process evidence
- ISO

The first two activities were discussed in detail in Section 2.

The evidence gathering/library building was an exercise requiring projects to produce specific evidence for key aspects of project processes. There were several benefits to this exercise:

- It allowed the process engineers to review evidence and point out potential deficiencies (so projects could make adjustments)
- It disciplined the projects into reviewing just how processes were being implemented.
- It enabled the sharing of concepts across projects through the sharing of artifacts and the discussion of approaches at process deployment team meetings.
- It helped to identify processes that may be misused or ineffective.

ISO was almost universally identified as one of the most beneficial tools adopted in pursuing excellence in process within SEAS. Although CMM had been part of the culture within the organization for over 7 years, the use of ISO was identified as one of the top activities in attaining excellence. Several reasons were given for this:

1. ISO addressed the entire SEAS organization as opposed to software projects and personnel only. This required that all personnel be involved in the concept of process which resulted in SEAS becoming a fully integrated enterprise with process as a major theme.
2. ISO was much easier to understand and to adopt than the full suite of CMM KPAs. It de-emphasizes process detail and focuses on understanding and applying the basics.
3. ISO successes gave the organization a 'can-do' attitude which was reflected in a much higher level of confidence when more detailed reviews of CMM were addressed.

Section 4 Lessons Learned

As was noted previously, detailed records of the experiences, costs, impacts and general impressions of the overall activities were archived by the process improvement team. In reviewing this information and by carrying out extensive interviews with project personnel and managers, the successes and shortcomings were analyzed in an attempt to identify the most effective activities and approaches that led to the high maturity level of the SEAS Center. There are 7 points that were gleaned from the experiences as reflecting the most important activities that an organization should adopt as part of their improvement program.

Recommendation 1: Operate as a Level 5 Organization

This recommendation suggests that an organization should not focus on sequentially addressing the CMM Levels from 2 through 5 nor should they focus on sequentially addressing individual KPAs. Instead, the most important element of the improvement program is to establish a culture of continuous improvement based on the goals and needs of that organization. The concept of 'continuous improvement' can be termed an 'optimizing' organization (Level 5) and has several key elements that should be established from the start:

- Focus on improvement of the product (as opposed to merely improving process). Such goals as cutting defects or improving productivity or decreasing cycle time should be the measure of change; not the number of processes that are established.
- Step 1 is to define the baseline of the products and process. That implies that the current product characteristics (cost, time, defect rates, effort distribution, etc.) must be captured along with the baseline of process characteristics (extent to which KPAs are satisfied). In addition to establishing the existing strengths and deficiencies of processes (via process gap analysis) one must generate a baseline or profile of the product characteristics. This information is the first step toward producing quantifiable information of the environment and is used to track impacts of process changes as well as to produce engineering models of the environment. Information for this baseline is collected from existing measurement data, surveys, project archives, interviews, and any other source of data that may provide some insight into the overall product profile.
- A measurement program is a requirement at the start of the overall improvement program. Some models imply that a mature measurement program may not be a critical element of early stages of an improvement effort, but the concept of operating as a Level 5 requires that a measurement program be established immediately. The measurement program is required for 3 specific reasons: (1) to establish models of the environment, (2) to manage projects, and (3) to guide change. An example of basic models generated early in the improvement program is depicted in Figure 4-1 The early data from SEAS was used to produce these models which in turn are used by managers and by process engineers. Such models can be generated very early in the

program and then may be continually refined as improved measurement data is collected.

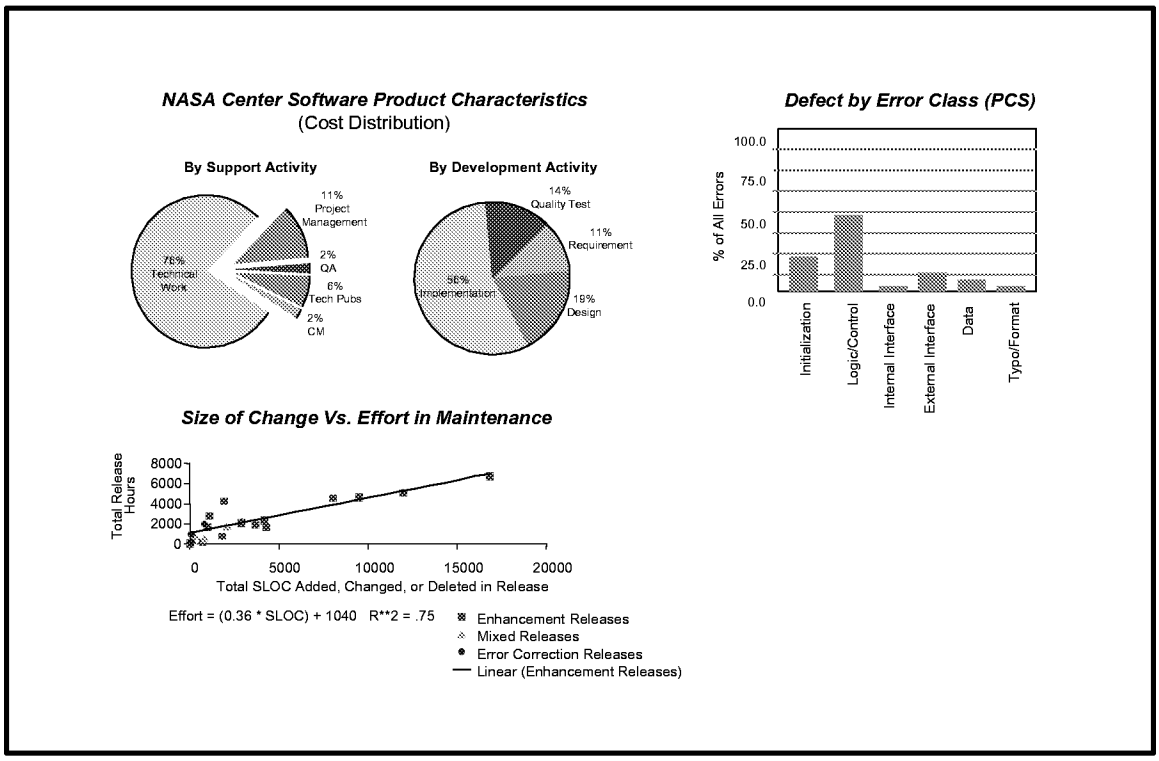


Figure 4.1 Sample Engineering Models of Process

- Both technical and management activities should be part of the improvement activity- as opposed to management only. Not only are process attributes important to the improvement program, but the selection and understanding of changing technical activities must be integrated into the program. This implies the continual infusion, tailoring and measuring of technical changes.

Recommendation 2: Set Specific Incremental Gates

Although the improvement program is viewed as a continuous, sustained program that has no completion criteria, incremental check points for the organization were a tool that accelerated the improvement efforts and acted as a catalyst for the program. These check points were most effective when they were performed by external reviewers; specifically SCE teams or ISO teams.

In the period June 1994 through November 1998, seven independent reviews were conducted. Obviously one has to be cautious of overtaxing the development and project organizations by requiring excessive time in participating in reviews, but the periodic reviews do act as a vital tool in assuring that all personnel are reviewing their adherence to processes and their awareness of the overall plans and goals of the organization.

Internal audits should be part of any organization's process program, but they do not replace the value of the reviews carried out by an independent, external team.

For the SEAS organization (about 850 persons) there were formal reviews occurring approximately every 6 months, sometimes more frequently. ISO surveillance audits occur each 6 months and the external CMM assessments occurred approximately yearly.

Recommendation 3: Adopt the Concept of 'Separation of Concerns'

Another critical element of a successful improvement program is that of organization. Not only must there be strong support from senior management, but there must be a designated process improvement organization whose responsibilities include expertise in process models, CMM, ISO, process improvement concepts, measurement and available assets within the organization. With one organization focusing on the concepts of process improvement and focusing on the generation of Program-level assets to be used by projects, then projects can focus on the task of producing systems and software.

In an 'Experience Factory' (Reference 8), one organization (PEO) is responsible for driving process improvement while the other organizations (projects) focus on the task of producing a quality product. It is not necessary that a project organization become expert in process models; it is only necessary that they work with the process organization in sharing information and adopting processes and assets made available to them.

The 'separation of concerns' concept implies that the project personnel are experts in producing systems and the process organizations are experts in process improvement and associated activities. There is no need to train project personnel in the details of process models such as CMM or ISO, it is only necessary they understand, and apply the process assets provided by the process organization.

Recommendation 4: Deploy Processes to Projects

One of the most effective steps in attaining process maturity was found to be that of having the process engineers work directly with the projects in helping to define, apply and understand appropriate processes for their particular project. This activity is in contrast to that of having the process staff work on writing, refining, tailoring, enhancing written processes. The effort put forth in working directly with projects will be much more effective than generating additional written standards.

Obviously there must be a written foundation describing the processes that are to be applied in the organization, but our experiences indicated that occasionally excessive effort is put forth in developing and refining written processes. The means by which the process engineers accelerate the 'deployment' of the appropriate processes is through the activity of 'shepherding' where process and quality engineers become experts in the organization's baseline, then they provide services to the projects in explaining just how to tailor, implement, and sustain relevant processes on their projects.

In addition to the shepherding activity, the process engineers should adopt the idea of scheduling periodic (weekly on SEAS) 'Process Deployment Team' meetings where a 1-hour discussion of process implications and use is presented. All managers of the organization are invited and the process engineers lead a discussion of a process topic; for

example 'How is the Quantitative Process Management KPA applied on a project in this domain?' or 'What engineering models of the environment exist for our use and how do we use them'?'

It is the responsibility of the process engineers (SEPG in CMM terminology) along with the Quality Assurance office to provide services to the project organizations by identifying appropriate assets for the projects and to help them apply these assets; without burdening the projects with undue overhead.

Recommendation 5: Measure Improvement by Product Not by Process

There is the commonly accepted belief that the quality of the software product generated is directly affected by the processes used to generate the product. For that reason, organizations implementing a process improvement program, in reality are targeting to favorably impact the end products generated by the development. They are anticipating improvement measured by product measures, ie., cost, defect rates, cycle time, accurate estimation, etc.

Although this is an obvious and simple concept, organizations occasionally overlook the importance of continually tracking the end product to verify that improvements in process are meeting the goals of improving the product. Too often, we measure success as the attainment of certain CMM levels, or ISO registration or producing more extensive processes. Measuring and tracking the product change is often overlooked. Although it is very difficult to measure trends in products over a long period of time, the exercise of establishing goals, defining measures, and capturing the starting point of these measures is valuable in itself. It provides the discipline of understanding the projects and understanding the environment through the generation of models, goals, and applied measurement.

Senior managers as well as clients often pose the challenge of proving the worth of the process improvement program. Instead of arguing that these people '…just don't understand the value of process…', the process organization must be prepared to respond to such challenges with specific measures that represent the product; not only the process. The questions are very appropriate questions and the measurement program must concentrate on continually capturing product attributes so that such questions can be addressed; even when the results may not show the expected benefits of the program.

Recommendation 6: Allocate Appropriate Resources

The activity of process improvement as well as process in general, requires effort. Although the goal is to have the process improvement activity produce a greater return on investment than the cost of the investment, the overall activity still requires a sustained effort. It is recommended that any organization identify the level of resources that it will commit to sustain the processes and process improvement program, then adhere to that commitment as it would with any project. It is a mistake to assume that this activity can be absorbed as 'no cost' by merely requesting that project personnel devote several hours

per week on the activity and that specific resources do not have to be allocated. From the experiences at SEAS, this approach will not adequately support the process program.

Based on nearly 8 years of experiences with varying size of organization, it was found that the typical allocation of resources for the process program was approximately 1% to 1.25% of the size of the entire organization. This effort is in addition to the specific project activities that will require additional resources. It also is recommended that the Quality Assurance activities allocate from 1.25% up to 2% of the organization that it is supporting.

Table 4-1 shows the relative cost of the process activities for different size organizations. The data is based on direct experiences of SEAS over the 8 year period.

- Requires .8% to 1.3% for process improvement activity
- Quality Assurance requires from 1% to 1.5%
- Spend 2 to 3 times more effort deploying versus writing processes

Program Size	0-20% Software	20-40% Software	40% Up
70 - 150	1.5 FTE	2.0	2.5
150 - 400	2.0 -2.5	2.5 - 4.0	3.0 - 4.5
400 - 900	3.0 - 4.0	3.5 - 4.5	4.5 - 6.0
900 - 1700	3.0 - 5.0	4.0 - 6.0	5.0 - 7.0

Table 4.1 Allocate Appropriate Resources (Based on SEAS History)

Recommendation 7: Produce 3 Specific Documents Early

There are numerous activities that must be addressed when an organization initiates a process improvement program and there are several products that also must be considered. Based on the SEAS experiences, it is recommended that 3 specific documents be produced or at least planned when the process program is established.

The 3 documents include: (1) Quality Management System (QMS) document, (2) process improvement plan, and (3) profile of the organization.

1. The QMS is a required document of ISO-9001 and has proved to be an extremely valuable handbook for SEAS as well as other organizations who have produced such a document. It has been used as an orientation guide for new employees and is a valuable reference for all personnel in characterizing the business operations of the program. It is recommended that the document capture:
   - Description of the organization and the staff (roles and responsibilities)
   - Description of the processes in place including their application.

     Standards, policies, methodologies, handbooks and general guidance.
   - Overall process planning (measurement program and process improvement program)
   - Description of how the organization complies with required benchmarks (ISO, CMM, SA-CMM, etc.)
2. The Process Improvement Plan (PIP) describes the goals, responsibilities, and approach to attaining the improvement goals. It adds the structure of a project to the activity with schedules, milestones, and most importantly- specific goals. The goals should include product as well as process goals.
3. The 'Profile' of the organization captures the general state of process usage by carrying out some type of gap analysis, but the bulk of the document should contain the product characteristics. This is the first step toward the goal of engineering software by producing quantifiable information. Sample recommended product information includes:
   - Amount of software in development and in maintenance
   - Distribution of effort across the life-cycle phases
   - Typical staffing profiles
   - Defect characteristics (number, type, severity)
   - Testing profiles
   - Maintenance costs/ per size of unit
   - Typical software cycle times (time to develop per size, time to make changes)
   - Variance in initial estimates vs. final actuals (size, cost, schedules)

Section 5 Conclusion

Over a 5-year period, the CSC SEAS Center carried out an aggressive process improvement program that resulted in an optimizing culture throughout the organization. The CMM Level 5 rating, achieved November of 1998, verified the success.

Focusing the success of the process improvement program on specific product goals, and using the compliance with industry benchmarks as a tool has helped make process improvement part of the SEAS culture. The QIP of the Software Engineering Laboratory (SEL) was used as the model for improvement and other industry benchmarks served as tools in achieving documented product goals. This paper describes aspects of the process improvement program that were key factors to the successful achievement of the CMM Level 5 rating.

The value of the investment made in process improvement was shown to be significant for the overall operations of the Center as well as the business opportunities. The quantitative value on product improvement was shown to be very difficult to determine and no conclusions could be made there.

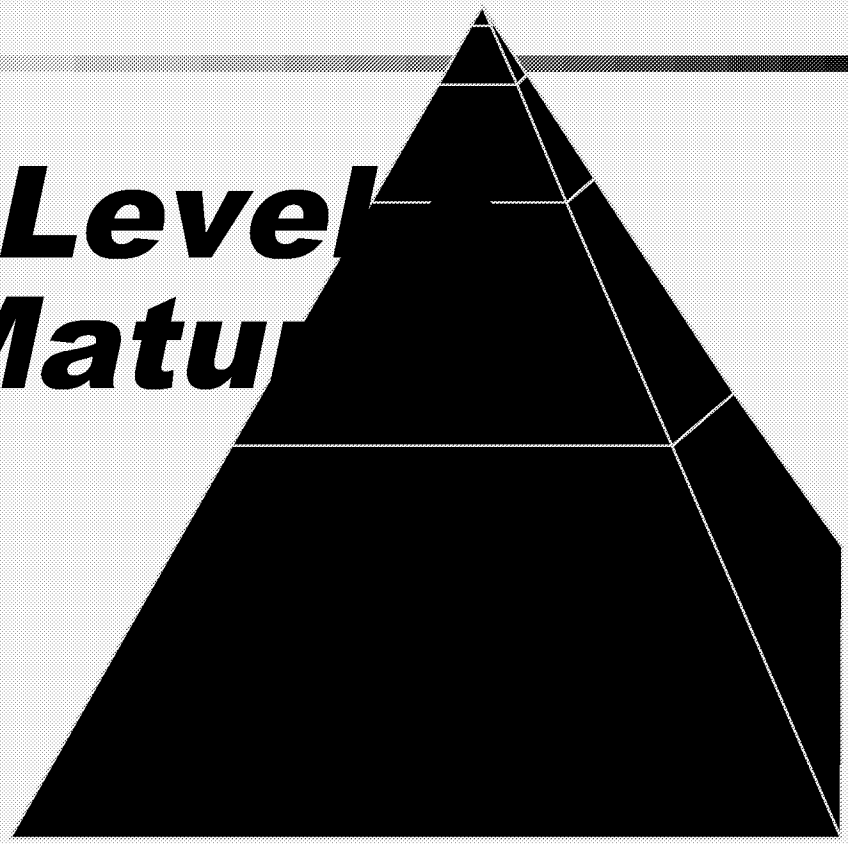
As a result of the five years of activity, the SEAS Center produced seven recommendations that any organization should follow in implementing a process improvement program. These recommendations focus on building a culture of continuous change and improvement throughout an organization.

References

1. Paulk, M., Curtis, B., Chrissis, M., Webber, C., "Capability Maturity Model for Software Version 1.1", CMU/SEI-93-TR-24, February 1993
2. Basili, V. R., "Quantitative Evaluation of a Software Engineering Methodology," *Proceedings of the First Pan Pacific Computer Conference,* Melbourne, Australia, September 1985
3. F. McGarry, G. Page, V. Basili, et al., *An Overview of the Software Engineering Laboratory*, SEL-94-005, December 1994
4. CSC (internal document), "SEAS Software Profile", September 1995
5. CSC(internal document), "SEAS Process Improvement Plan", Version 3, April 1999
6. CSC(internal document), "SEAS Quality System Manual", Version 3, April 1999
7. McGarry, F., Decker, W., Burke, S., "Measuring the Impact of Process on Product", *Proceedings of 22nd Software Engineering Workshop*, December 1997
8. Basili, V. R., "Software Development: A Paradigm for the Future (Keynote Address)," *Proceedings COMPSAC '89,* Orlando, Florida, September 1989

24th Annual Software Engineering Workshop

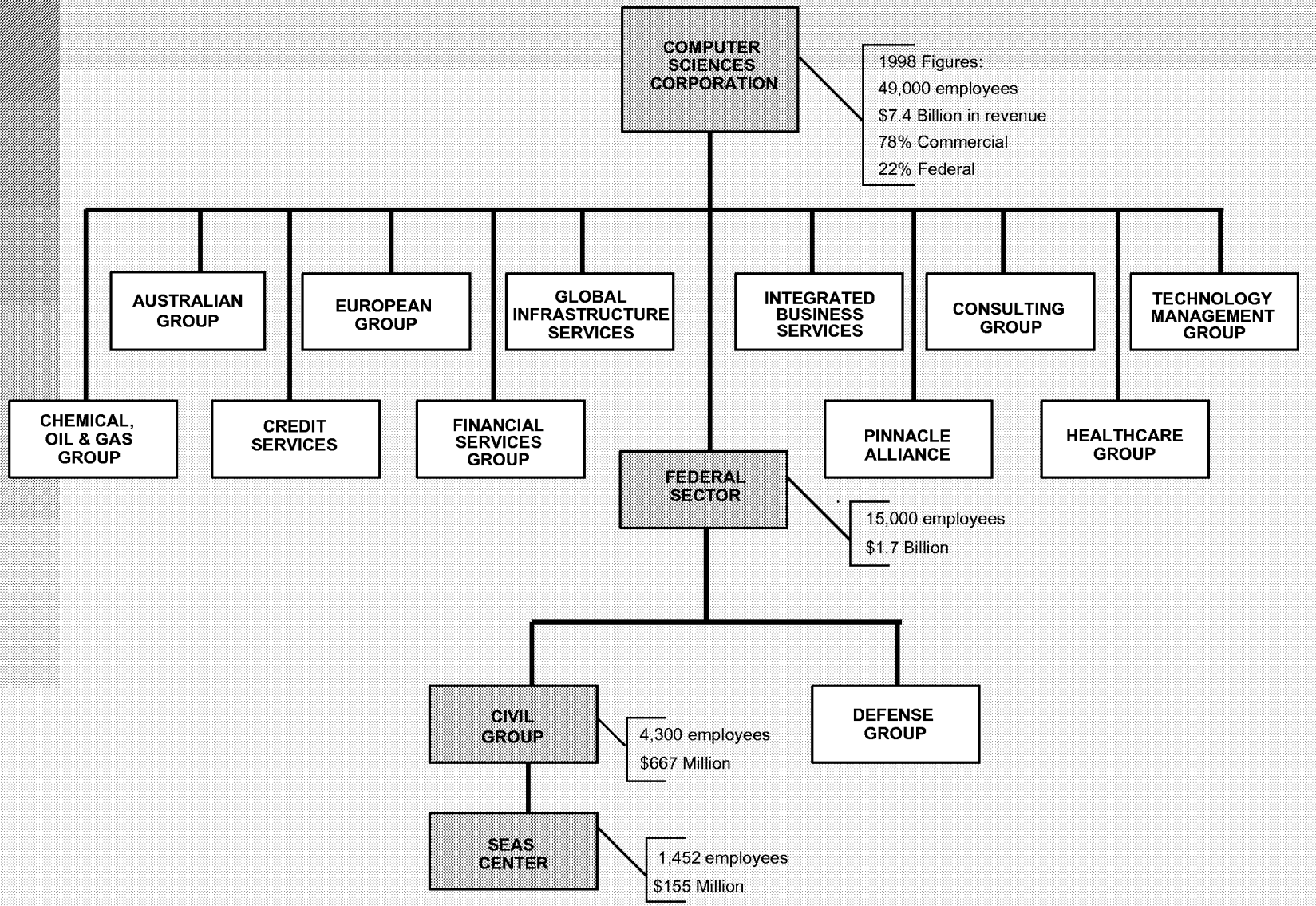
Attaining Level Process Matu



Frank McGarry
Bill Decker
Joe Haskell
Amy Parra

Computer Sciences Corporation

SEAS Center Within CSC



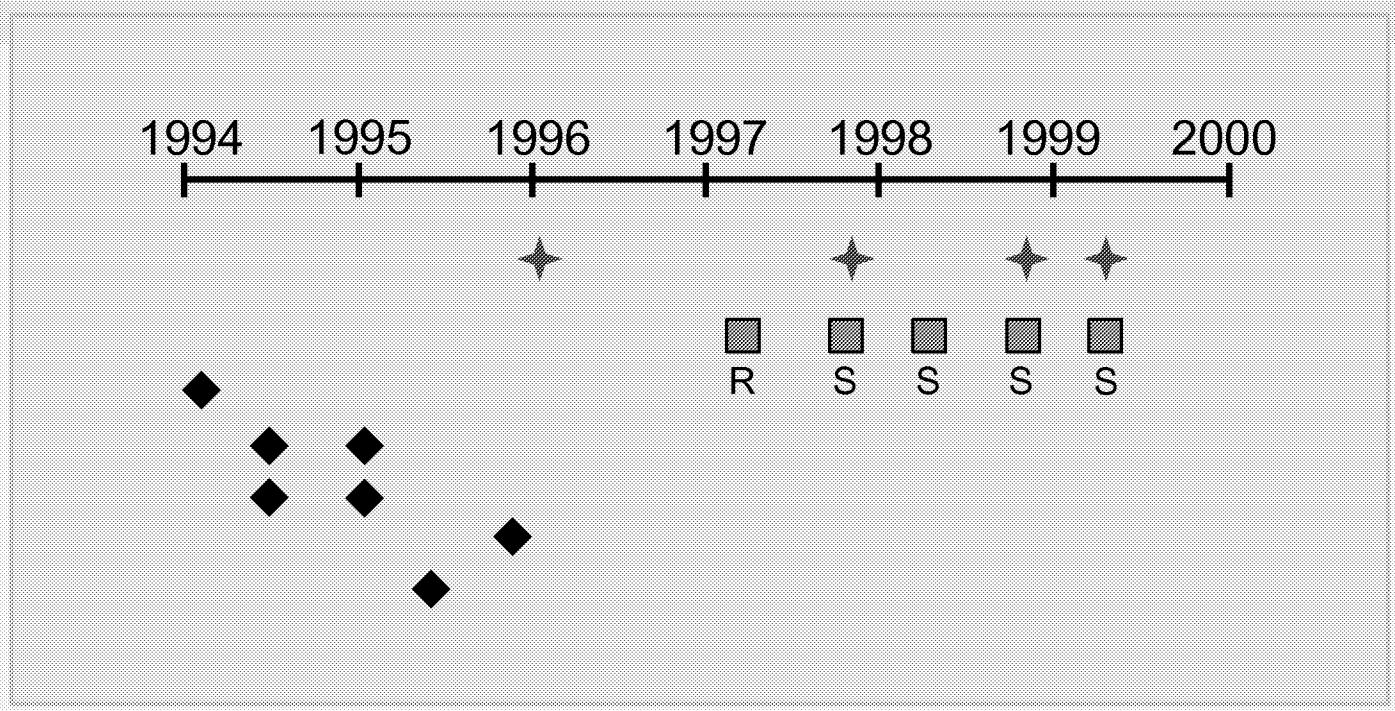
SEAS Center Process Improvement

- 1995 SEAS Process Improvement Plan specified six measurable goals
  - 5 product goals include product quality, productivity, predictability, cycle time, and technology infusion
  - 1 process goal; to attain full compliance with standard industry benchmarks
    - ISO-9001 Registration
    - CMM Level 3 (as determined by independent Software Capability Evaluation (SCE))
- In May 1997, registered to ISO 9001 standard
  In November 1997, assessed as CMM Level 3 and
  In November 1998, assessed as CMM Level 5
  In May 1999, reassessed as sustaining maturity
- Records of investment, changes, approach, and impacts were recorded to benefit CSC organizations and share experiences with professional community

SEAS Center Profile (1997-1998)

- SEAS Center has 850 personnel, primarily working on a NASA contract
  - 300 in software development and maintenance
  - 550 in systems engineering, analysis, and operations
  - Staffing has ranged from 700 to 1700 over past 10 years
- 20 software projects at any one time
- 10-year legacy of process improvement focus
  - First SCE in 1991
  - ISO 9001 registration in 1997

SEAS Center Benchmarking History



- SCE
- ISO 9001 registration audit (R), surveillance audits (S)
- Software process self assessments (SPA) and software process audits

Summary of Benchmarking Activities

	SCE (2/96)	ISO (5/97)	SCE (11/97-1/98)	ISO (11/97)	ISO* (6/98)	SCE (11/98)	Part of 'Group' SCE (5/99)
Preparation time	• 4 months	• 12 months	• 2 months	• 6 months	• 1 month	• 4 months	• 1 month
Organization effort	• 1800 staff hours	• 3400 staff hours	• 800 staff hours	• 500 staff hours	• 200 staff hours	• 800 staff hours	• 60 staff hours
Use of external consultants and training	• None	• 200 hours - Consultant - Internal auditor training - Pre-registration assessment	• None	• None	• None	• 2 staff days	• None
Preparation strategy	• Perform gap analysis • Use lessons learned from 1991 SCE • Focus on deployment	• Develop implementation plan • Use external experts • Train staff • Focus on deployment	• Complete actions items • Provide awareness seminars • Use internal assessments	• Continue process improvement initiatives • Focus on management review, internal audits, and corrective actions	• 1 refresher lecture • Internal audits continued	• Study/learn specific details of level 4-5 • Advance collection of evidence • Group seminars • Mock SCEs	• Update 'evidence' archives • Group seminars
Results	• 13 of 18 KPAs satisfied	• ISO registration achieved	• CMM Level 3 achieved Levels 4-5 (2 of 5 KPAs satisfied)	• ISO registration maintained (2 minor findings)	• ISO registration maintained (3 minor findings)	• CMM Level 5 achieved (18 of 18 KPAs)	• All reviewed KPAs satisfied

* Additional ISO assessments held in 11/98, 5/98, and 11/98

Experiences from the 4 year effort

- What was the cost?
  - Of attaining ‘process maturity’
- What was the ROI?
  - Benefits to business and organization
  - What impact does maturity have on s/w cost, quality, manageability
- Which activities were hardest/ easiest/ most beneficial/ least beneficial?
- What would I tell another organization to do/ not to do?
  - 7 lessons learned

Cost* Distribution for Process

For organization of 800 persons-over 4 years

Activity	4 year cost	1995-1996	1997-1998
Develop/Maintain Processes (write/update)	6 SY	40%	15%
Deploy/Training/Awareness	10 SY	10%	40%
Infrastructure (data base, libraries, distribution)	2 SY	5%	5%
Process Improvement (planning, studies, experiments, analyzing)	8 SY	15%	30%
Assessment Preparation (SCE, ISO)	3 SY	25%	5% - 5%
Reporting/Reviews	1 SY	3%	3%

* Includes cost of developing processes, deploying,measuring, training, maintaining (packaging), developing infrastructure, process improvement; does not include cost of project ops doing CM, QA, Planning, etc.- it does include their cost participating in studies, training, audit participation) Cost based on time: July 1994 through November 1998

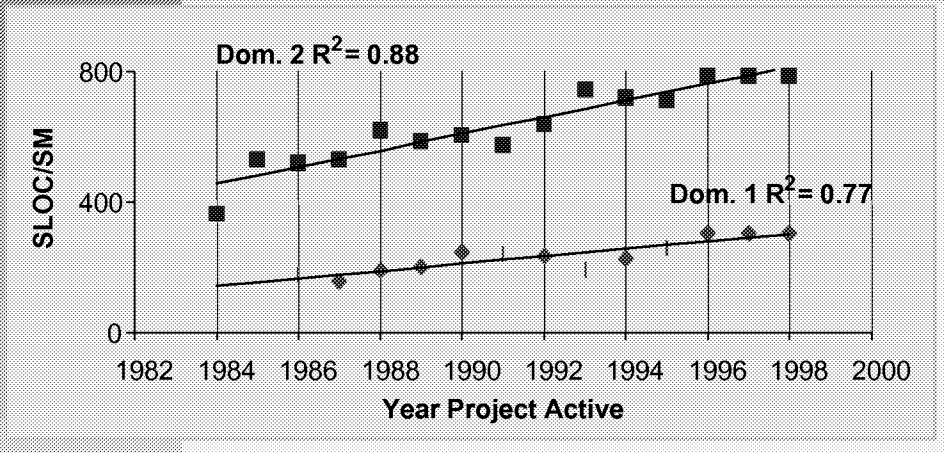
Return on Investment (ROI) from Process Improvement Efforts

- Organizational Value (SEAS) *
  - Drove technology and operational enhancements
    - Increased focus on achieving organizational goals (began operating as an enterprise)
    - Improved communication, teamwork, and understanding and use of organizational processes
    - Provided *discipline* for accelerating improvement programs
    - Resulted in updating of policies and processes to address real needs of organization
    - Accelerated adoption of technology across organization (e.g., electronic document libraries)
    - Fostered pride in achieving one of organizational goals
- Business Value
  - Major contributor to over $500.M new business (first year)
    - 3 major 1998-99 program awards driven by adoption of processes/experiences of SEAS
    - The corporation competitive position has been enhanced in numerous other offerings
    - Provided the first Level 5 Credential for CSC
    - Level 5 proven processes on-line for use by all of CSC

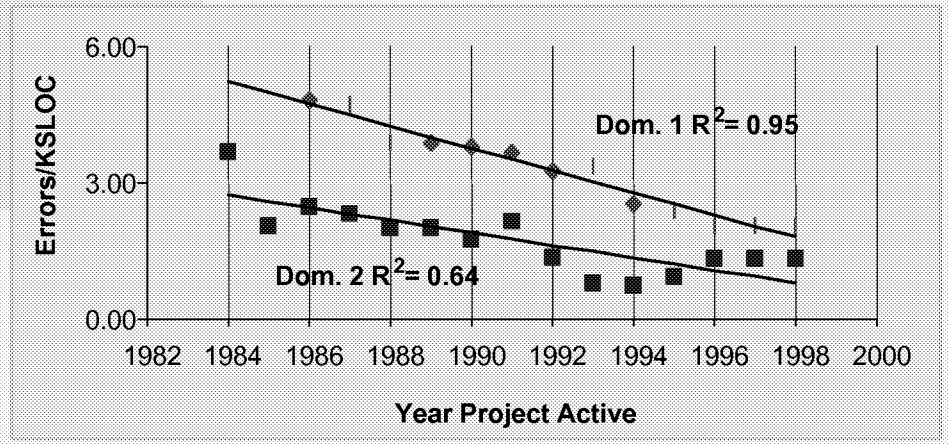
* Based on multiple 'Lessons learned' reports, employee surveys, and assessments by SEAS management staff.

Productivity-Quality Trends Over Time

Average Productivity for all Projects Active in Year

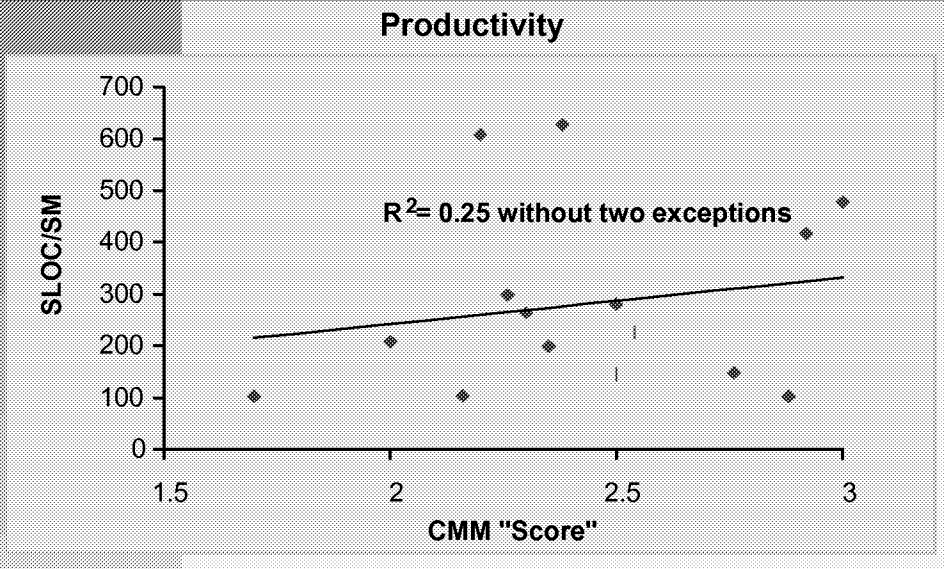


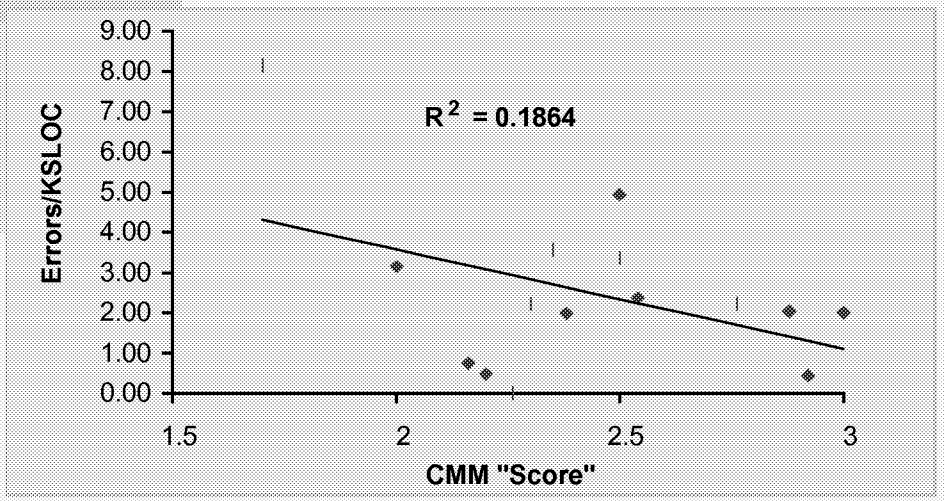
Average Defect Rate for all Projects Active in Year



- Consistent/constant 6% per year productivity improvement - even prior to CMM Level 1 rating
- Also 5% per year quality improvement - even prior to CMM Level 1 rating
- No change in improvement rates after aggressive improvement programs started

ROI: Impact of CMM Maturity on Cost, Quality, Manageability





X axis determined by number of KPAs satisfied, partially satisfied, not satisfied

- Detailed measurement data on 90 projects was accumulated (over 9 years).
- Data included accurate product data (cost, defects, size, etc.) and process data (based on assessments).
- Analysis to determine correlation between process maturity and product data showed minimal correlation.

Conclusion: Extremely difficult to measure process impact

Relative Impact of Improvement Activities (Most Significant)

- Shepherding
  - PEO and QA personnel assigned to work directly with projects in guiding process usage and development
  - Project personnel rated this as one of the 2 most helpful assets toward improvement
  - Separation of concerns (SETO vs. SEPG)
- Friday sessions
  - Weekly meetings run by PEO where all managers invited to discuss 'process'- took form of briefings and planning sessions by process/quality staff
  - The weekly process meetings instilled the concepts, goals, asset sharing that had never been attained previously
- Library creation/evidence gathering
  - Activity of producing evidence of processes and assets and capturing in common library provided 2 benefits:
    - Widely shared and actually used
    - Forced the refinement of assets that were '90% complete'
- ISO
  - Goal of only writing what is actually done and limiting the concepts to basics provided significant leverage

Relative Impact of Improvement Activities (Least Significant)

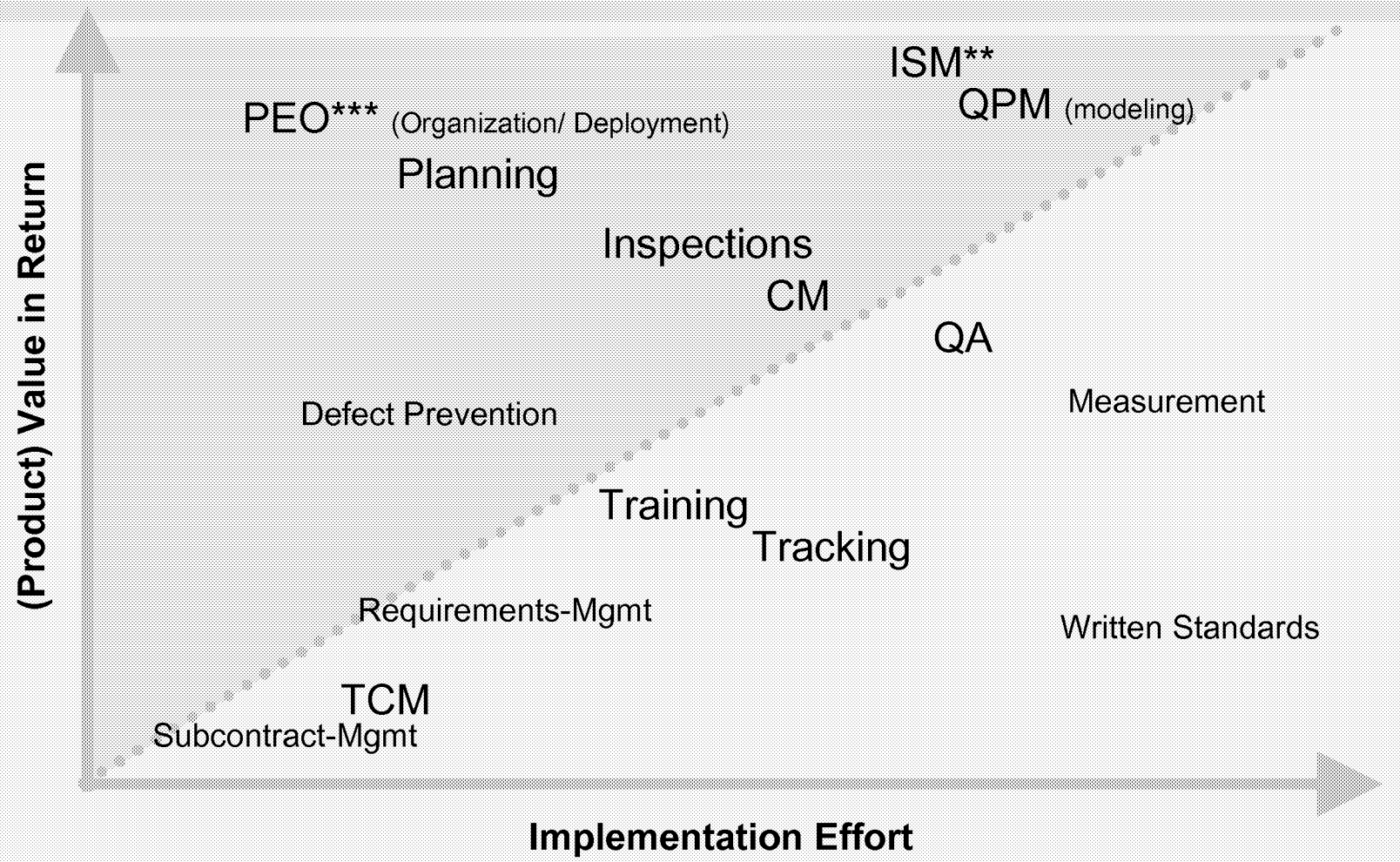
- 'Model driven' measurement
  - Measures collected because they are in model/ as opposed to part of goals. (GQM)
- Compliance matrices/ Gap analysis
  - Exercise of continually mapping perceived weaknesses and strengths to some model matrix required major effort- yet results were rarely used to improve local process
- Writing/ refining/ tailoring standards
  - Nearly 40% of process effort in '94-95 was spent on writing, rewriting standards - as opposed to deploying concepts and approaches to projects. The most used written word can be found in relatively small number of the pages
- Training in CMM, ISO, SA CMM
  - Details of the process rating models are of very limited value to the project staff. The effort we put into training personnel in these models had very limited value. We should have been focusing on the concepts of process and improvement.

ROI* for Process Activities

(Investment is cost of establishing- not operating)

(Return is benefit to projects and organization)

In I Environment- SEAS



*Results based on effort to implement, audit reports, and project evaluations

** ISM includes PAC process and process handbooks

*** PEO includes shepherding, deployment training, PI

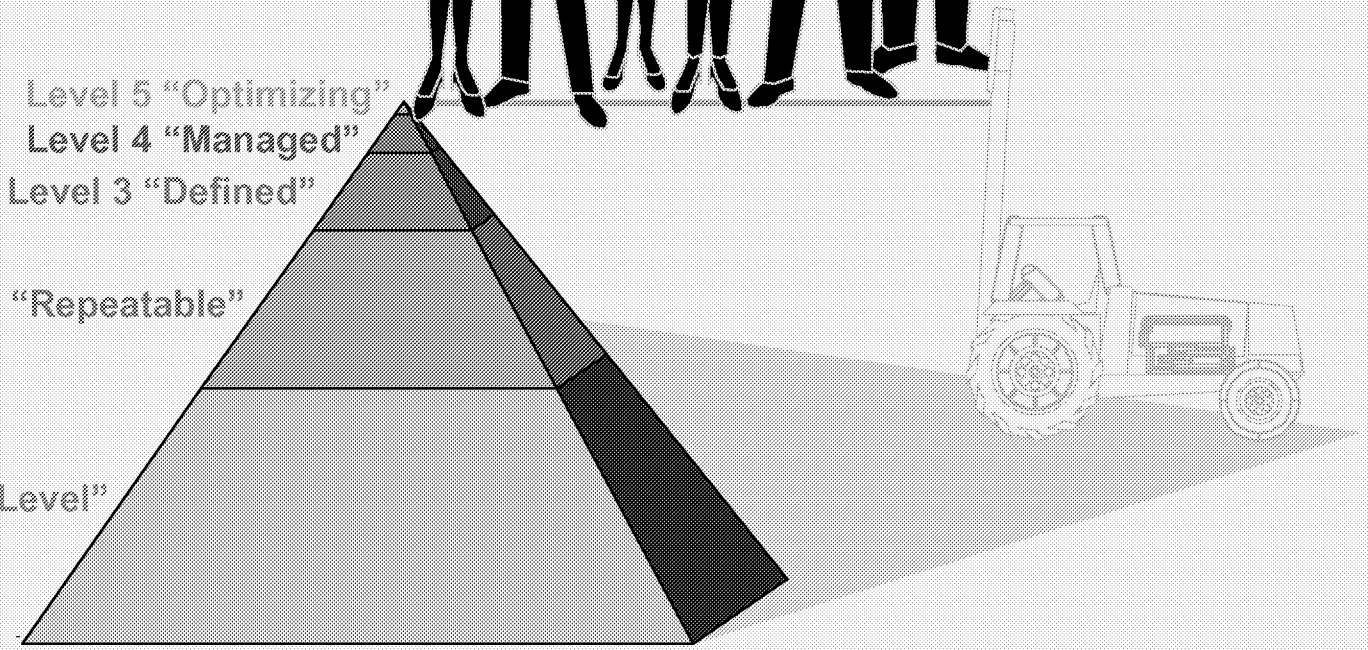
7 Steps to Success

Based on SEAS Process Improvement Experiences

1. Operate as a level 5 at start
2. Set Specific incremental ‘gates’
3. Adopt concept of Separation of Concerns
4. Deploy processes to projects
5. Measure improvement by ‘Product’- not by ‘Process’
6. Allocate appropriate resources
7. Produce 3 specific products early

1 Operate As a Level 5 Organization

- Goal is to improve product
- Baseline is process and product understanding
- Many goals drive c
- Change based on
- Measurement is fu
- Emphasis is on tech ca d ge ties

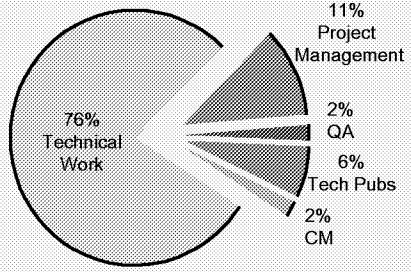


Measure from 'day 1'; Build models to 'understand'

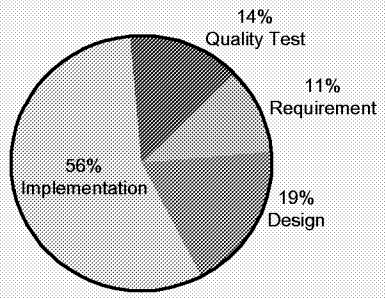
Engineering Models of Processes

NASA Center Software Product Characteristics
(Cost Distribution)

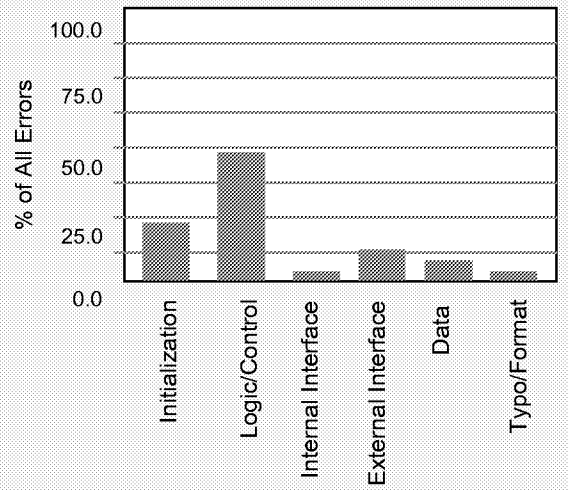
By Support Activity



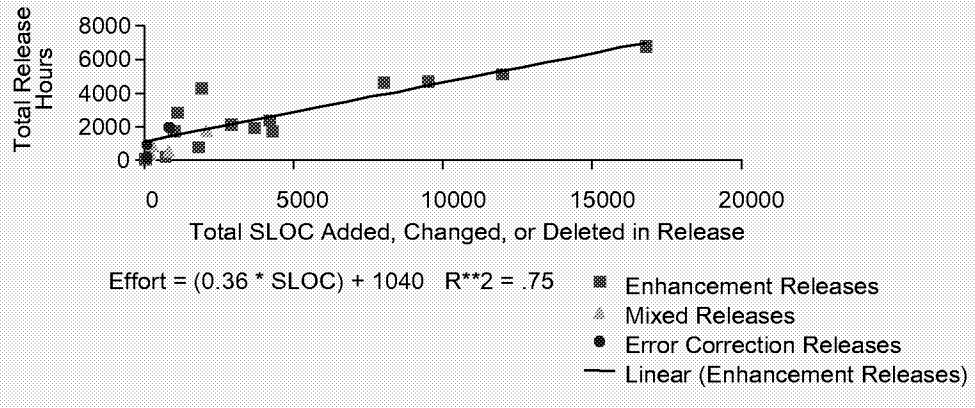
By Development Activity



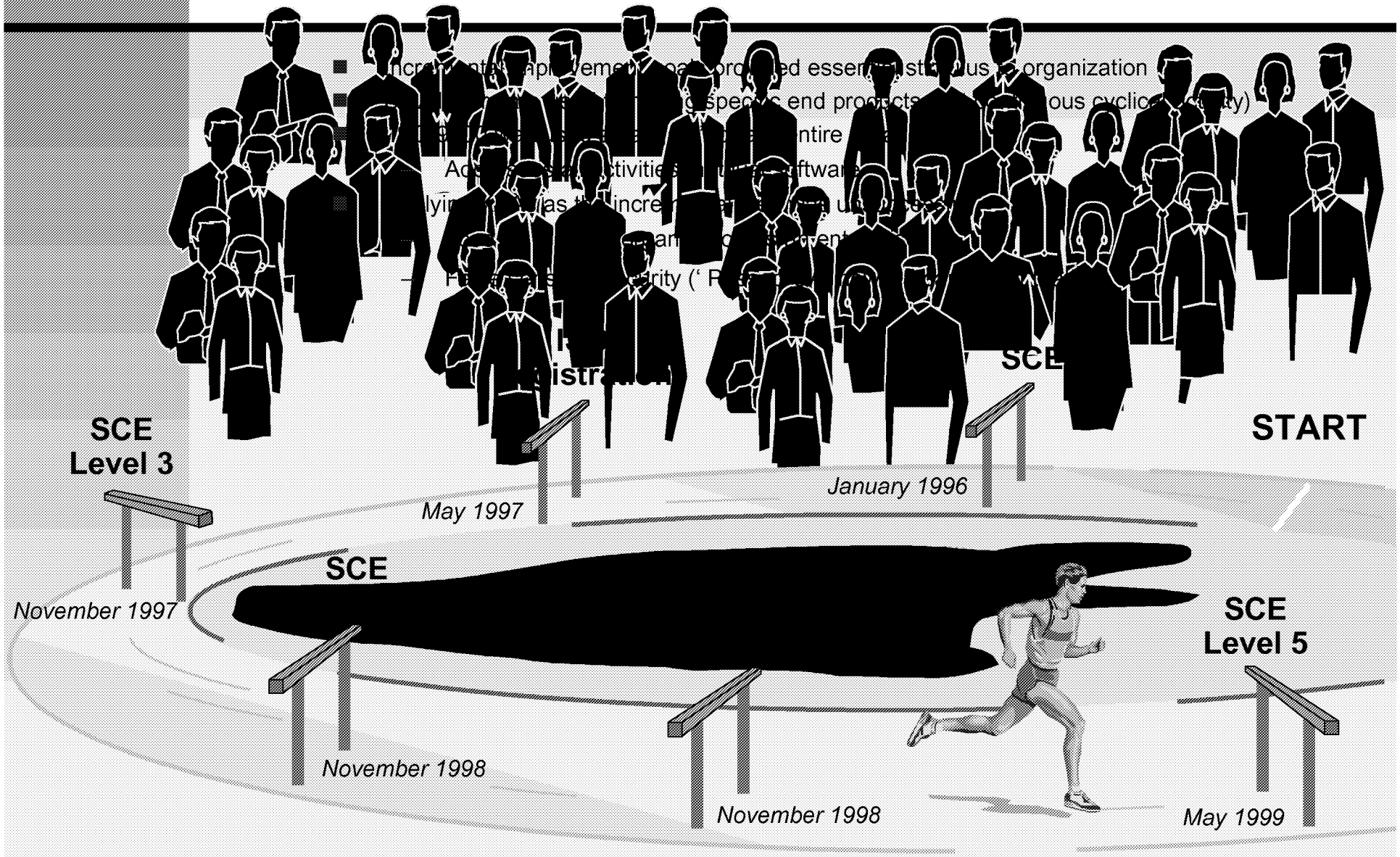
Defect by Error Class (PCS)



Size of Change Vs. Effort in Maintenance

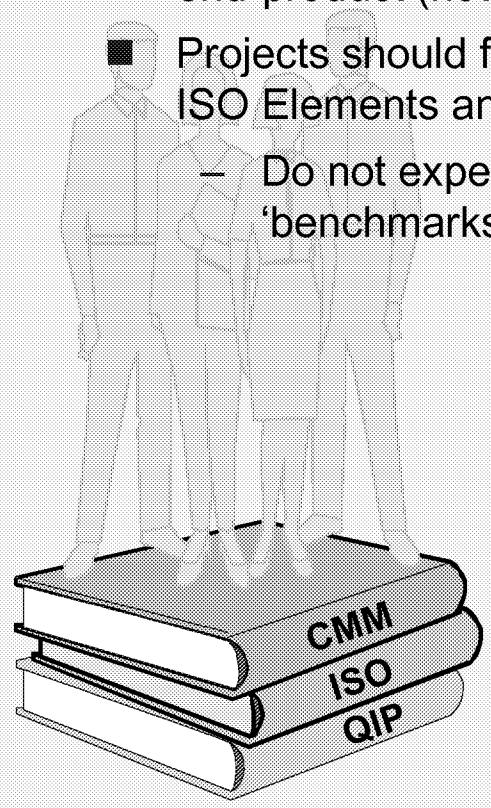


2 Set Specific Incremental "Gates"

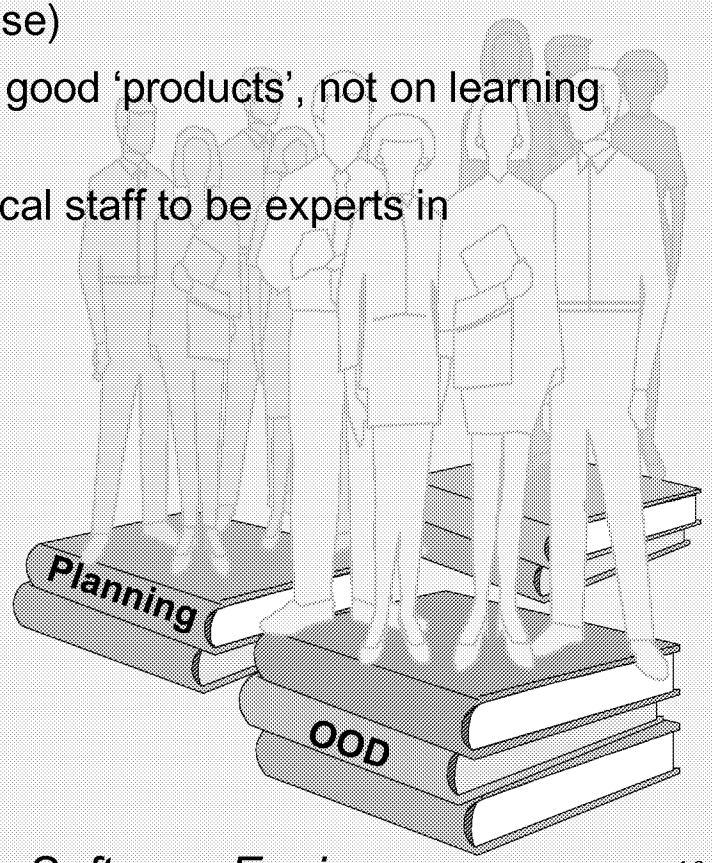


3 Adopt Concept of Separation of Concerns

- Hide details of benchmark requirements
  - Training must focus on needs of the project, not on detail of CMM, ISO and standards details
- Measure success of projects by ability to produce end-product (not by process expertise)
- Projects should focus on producing good 'products', not on learning ISO Elements and CMM KPAs
  - Do not expect or require technical staff to be experts in 'benchmarks (CMM, ISO, etc.)



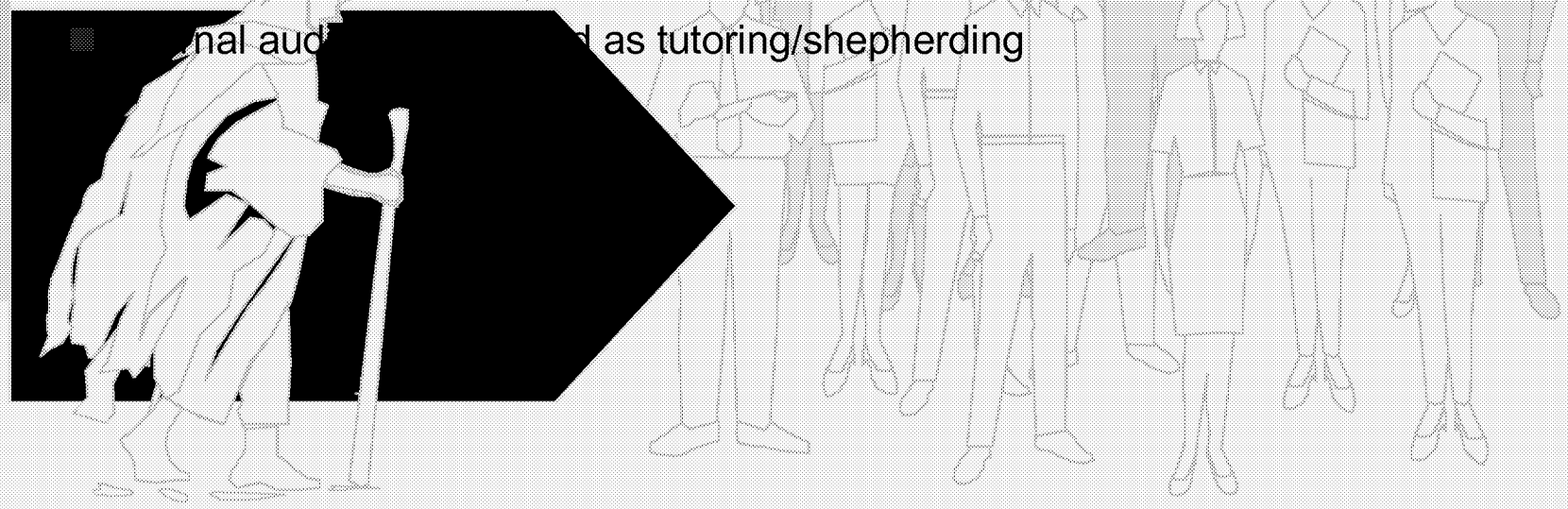
Process Engineers



Software Engineers

4 Deploy Processes to Projects

- Expend 2 to 3 times as much effort in deployment as writing processes
- Friday meetings
  - Combine concepts of ‘tutorials’, sharing (of project experiences) and project status toward reaching some gate (e.g. preparing for SCE, or ISO)
- Concept of Shepherds for projects
  - Process experts (from PEO and QAO) act as consultant to specific project
    - Process staff must be viewed as a service - not an overhead
    - Each project/task is assigned ‘shepherd’
    - In some fashion, similar to intent of SEPG
- [illegible]nal aud[illegible]d as tutoring/shepherding



Process Engineers ***Software Engineers***

5 Measure Improvement by "Product" not by "Process"

Start

- CMM level or benchmark level can provide false sense of accomplishment (or false sense of incompetence)
- CMM and ISO are great tools; they are lousy goals
- Processes should focus on improving your organization; not on complying with benchmark

Finale

6 Allocate Appropriate Resources

*Based on SEAS history**

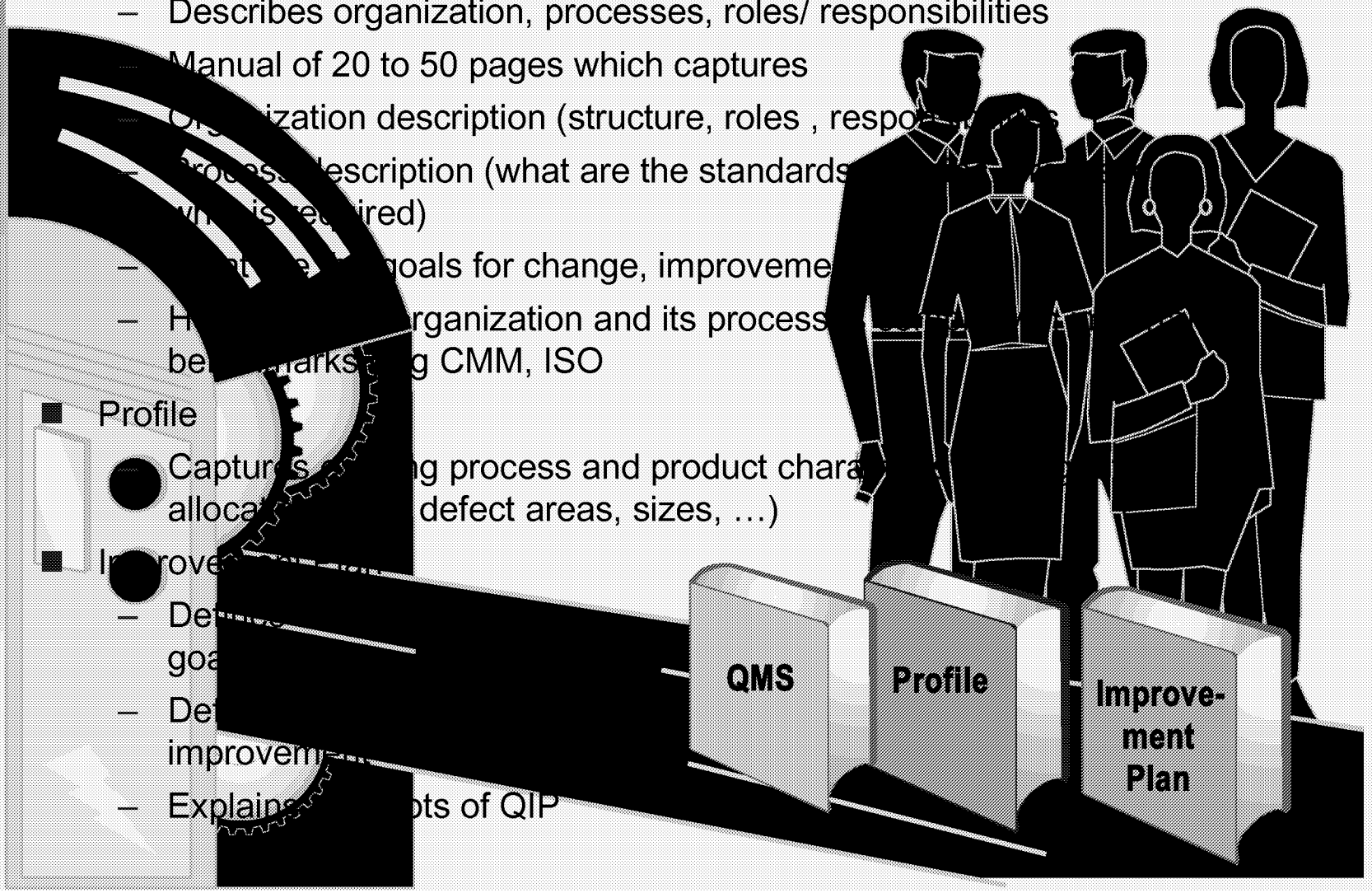
- Requires .8% to 1.3% for process improvement activity
- Quality Assurance requires from 1% to 1.5%
- Spend 2 to 3 times more effort deploying versus writing processes

Program Size	0-20% Software	20-40% Software	40% Up
70 - 150	1.5 FTE	2.0	2.5
150 - 400	2.0 -2.5	2.5 - 4.0	3.0 - 4.5
400 - 900	3.0 - 4.0	3.5 - 4.5	4.5 - 6.0
900 - 1700	3.0 - 5.0	4.0 - 6.0	5.0 - 7.0

**These are identified process personnel for "Program" - (Project support is in addition) (Does not include QAO)*

7 Produce 3 Specific Products Early

- QMS
  - Describes organization, processes, roles/ responsibilities
  - Manual of 20 to 50 pages which captures
  - Organization description (structure, roles , responsibilities)
  - Process description (what are the standards which is required)
  - [illegible] goals for change, improvement
  - How the organization and its processes benchmarks [illegible] CMM, ISO
- Profile
  - Captures [illegible] process and product characteristics allocation of defect areas, sizes, …)
- Improvement Plan
  - Defines [illegible] goals
  - Defines improvement
  - Explains [illegible] pts of QIP



Conclusion

- Process Maturity can be achieved as long as:
  - A structured approach towards the ‘process’ goal is followed
    - Actually the goal is ‘product’ improvement and must be measured as such (success is lower cost, not CMM Level 5)
  - There is a group within the organization that shepherds the improvement process
    - Senior, competent personnel
  - Focus remains on process improvement (vs. “just meeting the benchmark”)
  - Latitude is provided to support ‘cultural change’ for overall organization
  - There is management sponsorship and commitment (Resources, participation, ‘patience’)

Lessons Learned from the Failure of an Experience Base Initiative Using a Bottom-up Development Paradigm

Arne Koennecker, University of Kaiserslautern/Fraunhofer IESE
Ross Jeffery, University of New South Wales
Graham Low, University of New South Wales

Address for correspondence:

Ross Jeffery
Centre for Advance Empirical Software Research
School of Information Systems
University of New South Wales
Sydney 2052 NSW
Australia
Email: r.jeffery@unsw.edu.au

Abstract.
This paper describes the development of an experience factory in an Australian organization. Information structures were well developed and used in the daily work of the organization. This included the use of network technology as well as the personal interaction between department members. Highly motivated personnel drove improvement via new techniques, knowledge, and tools. A special focus existed to simplify work tasks through tool support. Daily work and problem solving was strongly based on personnel interaction and access to knowledge bases (documentation, mail lists, etc.). The goal of the project was to package personnel experience and best practices and provide an effective framework for access and integration. The system was decommissioned shortly after the completion of the project. The reasons for this are discussed.

1. Introduction

Faced with improvement needs, in 1998 the company started to put special attention on approaches to support improvement activities in a structured way. Like many organizations in the software industry, improvement aspects and strategy issues ranged from product quality and project management to the overall improvement of software engineering skills. Further local improvement aspects had been identified in software process assessment using the CMM (Capability Maturity Model from the SEI [1]) and the ISO 9001 standard.

At the end of 1998, a project was started in co-operation with The Centre for Advanced Empirical Software Research (CAESAR) to evaluate the Experience Factory (EF) / Quality Improvement Paradigm (QIP) [2] concept. The concept was to be evaluated as an approach to support local improvement activities and to be applied as an approach in the given environment at the R&D department. The aim was to find a suitable approach within six months and to start realization of benefits as early as possible.

The choice of the EF / QIP concept was motivated by several aspects. Firstly, it was seen to be a promising concept that had been the subject of research projects in the past such as PERFECT (ESPRIT III project, sponsored by the CEC [3]). Secondly, the concept had already been applied in other organizations such as the Software Engineering Laboratory at NASA [4] and Daimler Benz AG [5]. Thirdly, the EF/QIP concept reflects the state-of-the-art in the field of improvement approaches, and therefore is of interest to the company.

The focus of the project was guided by five questions.

(1) Where has the EF concept already been applied, and what have been the experiences with it?
(2) What are the important characteristics of the company's environment, and of the company's philosophy, which need special consideration?
(3) Is the EF approach applicable considering the environment specifics in the organization.
(4) If (3) is true: How has the EF approach to be tailored so that it fits the needs and characteristics of the organization?
(5) If (3) is false: How can an organization-specific approach be developed which considers EF principles?

Principles of the classical EF approach

The EF approach describes an organizational framework, which addresses the issues of product and process improvement in software development organizations by providing an environment for continuous improvement. The EF approach defines an environment for controlled experimentation, knowledge reuse, experience packaging, and analysis of the development processes. The improvement environment consists of two parts: the project organization (PO) and the experience factory organization (EFO). Each of these follows

distinct steps in the Quality Improvement Paradigm (QIP). The project organization's major aim is to deliver software products according to given requirements. The PO uses information to improve, say, the product quality, the project performance or the reliability of project planning.

The Quality Improvement Paradigm

The QIP is the main driving force for continuous improvement and is integrated in both the PO and the EFO. It is defined as consisting of six steps [6]:

1. *Characterize* the current project and its environment with respect to existing models and metrics.
2. *Set the quantifiable goals* for successful project performance and improvement based on the first step and the business and project specific goals.
3. *Choose the appropriate process* model and supporting methods and tools for the project and define a project plan, which considers the decisions and definitions made in steps 1 and 2.
4. *Execute the process,* construct the products, collect and validate the data, and analyze it to provide real time feedback.
5. *Analyze the data* and evaluate the current practices, determine problems, record findings, and make recommendations for future project improvements.
6. *Package the experience* in the form of updated and refined models and other forms of structured knowledge gained from this project. Save it in an *Experience Base* to be reused in future projects.

The PO interacts during the project with the EF organization (EFO). The EFO supports it with knowledge and experience gained in the past and provides feedback about the performance and quality of the current project while analyzing the data provided. The task of the EFO, besides support during the software life cycle, is to package experience gained during projects in a reusable form and to store it in an Experience Base (EB).

The interacting PO and EFO realize two feedback loops, a project feedback loop that takes place in the execution phase (support & analysis), and an organizational feedback loop that takes place after a project is completed (analysis & packaging). The second feedback loop changes or improves the organization's understanding of software development by packaging and reusing experience and making it accessible to future projects.

How to build and run an EF

To start an EF there are two possible approaches: a top-down or a bottom-up approach. That is proceeding from defining processes, structures, products, and responsibilities to collecting concrete experience data, or else collecting data and proceeding back up a similar hierarchy. Basili and McGarry [7] propose a top-down approach, which aims to define and establish the required elements before the improvement activities and the data collection takes place. This provides a guiding, and more or less stable structure and the time to focus on analysis of results and products rather than on integrating changes in the structure while working with them. Five key steps characterize the described top-down approach: (1) Obtain commitment, (2) Establish structure (3) Establish processes (4) Produce baseline (5) Identify potential changes.

The EF at the SEL-NASA

The Software Engineering Laboratory (SEL) was started in 1976 at the NASA / GSFC comprising three organizations: NASA / GSFC Flight Dynamics Division, University of Maryland (Department of Computer Science), and the Computer Science Corporation (Flight Dynamics Technology Group). Its goal was to understand and improve the software development process and products within the GSFC Flight Dynamics Division. In this environment the EF concept was developed and first published in 1985 by V. Basili (with a later version in [2]) as a concept based on the research and experience of the SEL. Since then the EF has been successfully applied in the NASA environment and used in more than one hundred projects dealing with different improvement issues and technologies. The experiences range from detected impacts through the use of EF on product and process attributes, to recommendations as to what to consider when establishing an EF.

The EF at the Daimler Benz AG

Software plays a major role in the product range at Daimler-Benz. Outside of the SEL, the Daimler Benz experience is the only other report directly related to the establishment of an EF in a practical development environment. Furthermore, they describe their experiences in the first year of the EF project, which was significant to our need to establish benefits in a short time period. Three separate projects formed the basis of analysis. Project A was in the aerospace domain with mainly in-house software development of large embedded systems and rigid real-time constraints. A measurement program had already been commenced. The goals were to make improvement efforts persistent and repeatable, project effort predictable, and to support technical reviews. The

initiative comprised two application projects and 2-3 people were concerned with EF activities. Project B involved small-embedded systems. The development changed from contractors to in-house in recent years. The goal was to build core competencies and clarify development questions such as how to keep software portable, and how to make sure that each planned function was implemented. Review techniques were identified as potential support for this. The initiative comprised 1-2 application projects and 2-8 people were concerned with EF activities. Project C dealt with large administrative software units for managing internal business processes. Software requirements were defined in-house, but the development was outsourced. The focus for the EF was quality assurance, especially in outsourced development. In this case the initiative comprised 3 application projects and 2-3 people were concerned with EF activities.

For projects A and B the company followed the top-down approach discussed above. The measurement of the baseline started several months after the EF initiative. This first stage consisted of the definition of essential EF structures, processes, roles, and products. They also decided in project B to assist technical reviews and collect related data to help solve current problems. This was done without defining structures, and is therefore seen as a bottom-up activity.

For project C, they decided to collect potentially useful data immediately after the definition of fundamental goals using a one-day workshop. The EF elements like processes, tasks, and product structure were only defined when demand for that occurred. This characterizes an evolutionary approach and is seen by the authors as a bottom-up approach. The main reasons to follow this approach were:

(1) "The immature practices needed to be improved rather quickly, but they did not require highly sophisticated analysis techniques or experience structure documents.
(2) Structures would not be stable anyway.
(3) People were the bottleneck. Effort needed to be concentrated on content first." [5]

The choice between a top-down or bottom-up approach was further influenced by the opinion that stable and mature structures are needed for a top-down approach.

The experiences to date which were of most interest to this project were:

(1) Pros & cons of a top-down approach: The definition of the EF elements in the top-down approach makes it easier for the EF participants to recognize the existence of the EF but provides less concrete early benefits for them. The approach can not be performed without a close connection between the EF and the processes that are in place.

(2) Pros & cons of a bottom-up approach: It may enable a swift realization of the EF results. Results are visible in a short time, but this effect cannot be planned and it is often hard to prove the usefulness beforehand, making visibility of the EF benefits more difficult.

(3) There are many sources of reusable experiences and measurement is just one of them, e.g., intermediate products (like a QA plan) are often seen to be more useful for reuse than concrete experience packages, even when their impact has not been analyzed.

(4) There were no problems in handling and structuring the data. Collecting data and qualitative experiences were the bottleneck.

The EF in the PERFECT project

The PERFECT project is an ESPRIT III project funded by the Central European Commission (CEC) and started in the early 90's. Organizations like Daimler Benz, Siemens, Q-Labs / Ericsson, and the University of Kaiserslautern / Fraunhofer IESE came together with the aim to find a more detailed and tailored approach for the introduction of an EF into organizations. The benefits seen for the approach used include explicit goal setting, focus on products, establishment of a separate organization driving the improvement program, and the tailoring of the activities to specific needs. This is a realization of the principles stated in the EF concept [2].

2. Establishing the EF Goals and Methods

The following points were seen as important in establishing the strategy that would be adopted in the organization.

- Arguments exist that the EF assumes a stable environment, but that this is not suitable for all companies. Their environments may be too dynamic because of short technology cycles. Some organizations argue that stable structures might hinder progress and innovation.
- The time aspect is a critical point. The time frame for first results seems long when following the top-down approach. This can cause problems maintaining participant motivation and management commitment.
- The present EF / QIP approach remains a general, abstract framework, which lacks explicit implementation guidelines and detailed experience reports which are needed in industry. The data that is available at the moment is either experimental or based on a long-term application.
- The EF originated from a scientific and government environment at SEL-NASA and

proved suitable after long-term application. Are the results transferable to software companies in general?

- The bottom-up approach trialled at Daimler-Benz seemed to work as did the top-down approach. There is no detailed data for a comparison of the results of the two approaches. The bottom-up approach brought earlier results. Is a bottom-up approach the better alternative when preliminary processes and an understanding of the environment already exist?
- The EF / QIP approach requires a high degree of experimentation to evaluate techniques. Some companies, especially large ones with R&D departments, have the resources to do that but is it feasible in smaller companies? Often improvement decisions and technology adoptions have to be made much faster than is possible by using pilot projects.
- It is not completely clear whether improvements achieved related to things like reuse and productivity, have their root cause in the introduction of EF concepts or in the successful application of technology. Would the switch to promising techniques such as OO without the introduction of EF have had the same effect?

Setting the project goals

Based on this analysis it was decided that: "The project aim is to develop tools and techniques to improve the speed and quality of software development and to enhance the transfer of process knowledge between projects and project groups." In the organization there were six improvement initiatives present: (1) process tailoring, (2) CMM and ISO 900x assessments, (3) personnel skill improvement, (4) company improvement strategy, (5) self motivated tool development and tool integration (innovative spirit), and (6) the measurement program.

Thus several improvement activities were already present and action plans defined from the results. What the organization needed was a framework to support and focus the related actions. Process tailoring and definition existed and were already applied in parts of the department. Further action was needed to spread them out across the whole department and reuse experience gained during the initial implementations. It was not the main goal to achieve a state such as CMM level-5, which was seen as a hinderence to the company philosophy which was to establish an environment which is reliable and repeatable but not an overly defined one. In the organization the developers initiate a great part of the improvement activities. They identify problems and possible solutions, take ownership and develop solutions in the form of tools or work instructions. This was to be supported and recognized. The present personnel skill improvement activities were to be supported as well as the team spirit and the overall interaction / communication. It was viewed that stable and fixed structures tend to hinder that. The company strategy and goals had been broken down into improvement activities at the project and development level. The project needed to focus and refine these (GQM). The existing measurement program was showing promising results and indicating new improvement items.

It was determined that a bottom-up approach could build on current measurement and initially defined processes and could immediately deliver data associated with known improvement issues. It would also give incentive to the desired tool development. Next we set out to determine whether environmental conditions would also support a bottom-up approach. The situation in each of the project teams is significantly different with respect to techniques and tools deployed. There was an identified need to identify best practices, to document experiences with them, and to support the transfer of knowledge between project teams. It was obvious that the concept of the experience base could help. The information access environment was focused on network technology. Every project team had an internal / external homepage to spread information, they had a project server with related documents, a central mail and document repository existed to get information around and to document daily experiences. Documents templates give the information an identical structure to improve readability and to ensure consistency of data. The mailing and posting repository (Microsoft Outlook) had proved its usefulness in recent years by giving a basis for discussions and to disseminate information. Motivated by the general improvement spirit in the organization, the usage frequency of this repository was fairly high. It was possible to consider using this already-existing, documented experience for an Experience Base (EB). But what was still needed was an effective access technique for the information stored, e.g., a search engine.

Both the normal daily work and solution seeking resulted in high interaction between department members. People were identified as having special knowledge regarding different development fields and the general attitude was to provide others with this knowledge when needed. From unstructured interviews with team members it was identified that it would be useful to package the experiences (daily work knowledge). Initial examinations of the amount of already documented knowledge in reports and mail archives showed that a basic knowledge & experience base already existed on the Intranet but was not yet efficiently usable. Because of the lean hierarchy in the department,

self-motivated improvement activities and the integration of developer opinions was encouraged and simplified. This leads to an environment that is driven dynamically by the team members.

To summarize, information structures were well developed and used in the daily work. This includes the use of network technology as well as the interaction between department members. Highly motivated personnel drive improvement via new techniques, knowledge, and tools. A special focus existed to simplify work tasks through tool support. The daily work and problem solving was strongly based on personnel interaction and access to knowledge bases (documentation, mail lists, etc.). The goal therefore had to be to package personnel experience and best practices and provide an effective framework for access and integration.

From these findings we were convinced that the organization should establish an improvement environment based on the EF concept, but that the appropriate approach was bottom-up.

We defined the EF concept for the organization based on five steps:
Step 1 ***collect*** experience and knowledge,
Step 2 ***publish*** the experience documents and provides an access framework,
Step 3 ***integrate*** experience in an environment were it is needed,
Step 4 ***analyze*** how the experience repository is used, and
Step 5 ***extend*** the structures of the improvement environment when the need occurs.

What is different to the classic EF approach & concept?

The main difference to the EF/QIP concept described in [2] and the concept we describe is in the overall philosophy. First we favored a bottom-up approach starting with providing useable experience from the beginning rather than spending time defining processes and structures for a top-down approach. Moreover, our approach places knowledge management and integration in the center to serve as a driving force for continuous improvement. This is quite different to the EF, which uses the QIP [2] and the GQM [8] as driving forces.

One main concept of the classical Experience Factory is experience generation and explicit experimentation with new technologies to evaluate them and to measure their impact on product and process characteristics. Our approach goes away from explicit experience generation, and focuses on gathering existing experience and supplementing it as it grows using access technology. Furthermore it is not based on the principle of gathering experience from experimentation. Rather the approach uses experience gained with software engineering techniques and new software technologies in the daily work rather than explicitly experimenting with new things. Experience transfer supports the growth of the experience inherent in the environment.

Another difference is that our approach describes how to start the implementation of the first cycle (gathering of existing experience). Our approach allows both the improvement structures and the development environment mature over time. As in the EF framework our experience management environment (EME) supports the documentation and storage of every-day experience. Further more both approaches give structure to establish a continuous improvement environment. The EME is seen to be more evolutionary and able to be adjusted to special needs. The EF gives a predefined structure to be established and therefore changes the existing way to do things.

Requirements for application

Due to the fact that this approach was motivated by environmental characteristics, there exist certain requirements for the application. If another organization intended to apply our approach it should check the following characteristics, which we see as minimal requirements.

- A highly used and developed network environment has to be present and integrated in the daily processes.
- Information repository structures need to be present in the environment, i.e. an Intranet structure using mail archives, project servers, document servers, etc. is needed.
- At least initial processes have to exist, which define when certain information has to be documented, e.g., meeting notes.
- For the documentation style, corporate templates should exist, which give information a common structure.
- Activities have to exist which serve to identify improvement needs outside the knowledge management focus, e.g. CMM assessment.
- There has to be a conviction that there exists a high amount of already documented experience and knowledge in the environment. The document could emphasize things like process documentation, reports, mail archives, web pages, etc.
- The staff have to be self-motivated to search for experience or knowledge.
- The staff have to be self-motivated to experiment with new technologies for the improvement of products and their skills. The

company philosophy should support this by encouraging the staff to do so, e.g., planing time for that and recognizing those activities and the results.

- The organization must have an attitude to let the developers drive changes influenced by strategy and improvement goals. What this also assumes is that there exists a company thinking rather than an individual focus.
- An organization needs resources to establish the concept framework and to maintain it while it matures and grows.
- Initial process and project environment definitions should be available, which build a context for experiences and which can serve as success story examples. At least one project environment should exist, which has documented experience with the introduction of a defined development process.
- The project management staff should be open to constructive suggestion concerning improvements to their development processes.

3. The organizational solution

The project was initiated by a senior manager with a reputation as a successful champion. Staff were involved in all aspects of the initial concept design and subsequent implementation. A project manager from the organization was assigned the task of overseeing the experience factory project. Other staff were involved via seminars, individual consultations and an experience factory website established as a result of requests from the first staff seminar. Thus it was with confidence that we embarked on the technical design and implementation of the experience factory in the organization.

Since we were convinced that the environment already contained a significant amount of documented experience (the mail archive contained around 8500 documents six months after commencement), we began by finding an appropriate technology to gather this experience and make it searchable. Using the tool we selected (Microsoft Site Server 3.0) we also had a framework to make the gathered experience accessible via a web site. Therefore we created a separate web page providing the interface to the indexes. This also provided the integration step into their daily work. When someone wanted to find existing experience about a task or general information from the environment they could now do that using the web page.

Evaluation of indexing tools

The first step in the evaluation was to define the requirements for an appropriate indexing tool. These were separated into two kinds of requirements. We defined requirements for a surface-evaluation, i.e. an initial evaluation to check basic functionality, and when a tool passed these we evaluated it against further interface and behavior related requirements. In Table 1 they are marked as RSx (surface) and RDx (detailed) requirements:

Table 1. Requirements

Requirement	Description
RS1	*Price*: the price of the tool shall be reasonable, preferably freeware.
RS2	*File types*: the tool shall be able to create an index of common files including Microsoft Office documents, HTML, PDF, MS exchange files.
RS3	*Interface type*: the tool shall provide a web-based interface to apply queries on the document index to find appropriate information. It shall at least be possible to redirect the query input and output from a web site to the tool and vice versa.
RS4	*Scalability*: the tool shall be scalable, i.e. the amount of indexed documents and users should not be limited.
RS5	*Gathering*: the tool shall be able to gather documents over a Microsoft NT network. Tools running on a Unix machine but able to access NT would also fulfill this requirement.
RD1	*Performance*: the tool shall be reasonable fast, e.g., search queries shall be answered in less than a minute, re-indexing shall be possible over a weekend.
RD2	*Maintenance*: the tool shall provide a mechanism to automatically update the search base (scheduled builds).
RD3	*Interface style*: it shall be possible to provide the user with a short description of query matching documents and to modify the style of the interface.
RD4	*Access rights*: The tool shall be able to give a user only access to those files to which he has access over the NT network.

Based on these results we decided after three weeks of tool evaluation to use Microsoft Site Server 3.0 (MSS) as the tool to gather and publish our environment, experience & knowledge documents, to build the base for our experience management environment (EME). The network environment (Intranet) was a Microsoft Windows NT network. Clients were running Windows 98, Windows NT, or Windows NT Server. Furthermore a couple of servers running Unix are connected and their file system can be accessed over the NT network from other non-Unix machines. The document sources are summarized in Table 2.

Table 2. Document Sources

Document source	File types	Information type
Mail Exchange Server	Mail format (exch)	Folders for past and current project information, technology discussions, reuse items, etc.
Project and department web server	HTML, Microsoft Office documents (DOC, XLS, PPT), Adobe Acrobat PDF, database files (SQL, Access)	Project documents ranging from code to process descriptions, general department information like administration tasks
Local workstations	Microsoft Office documents (DOC, XLS, PPT), Adobe Acrobat PDF, HTML, plain text (TXT)	Documents gathered for own information purpose, document drafts

4. Analysis of Usage

The analysis step consisted of preliminary analysis of the access log data to the web. The tool provided us with the functionality to create usage reports. In addition we conducted a survey on the benefits the people observed while using it. In three months we were able to implement the structures for the four steps 'collect', 'publish', 'integrate', and 'analyze'. We were able to identify items for extension activities (step 'extend') from these results and from user feedback, which helped to focus on the future.

The growth of our document search repository over time was influenced by three factors:

- including more document sources in a specific catalog,
- including more types of documents in a catalog definition, and
- the growth of experience & knowledge documents in the environment over time.

The number of documents changed with every build cycle for the catalogs. Numbers of gathered documents in the search repository have been documented and are shown in Figure 1, together with the factor which mainly influenced the growth. Here we see a significant growth in documents available after a relatively short period of time.

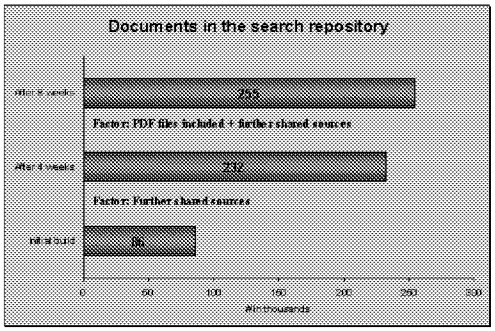


Figure 1. Growth of Documents in the Search Repository

Figure 2 shows the growth of the mail repository which results largely from daily work. The figure shows that in the last four weeks of the project the growth in the mail repository was 1,800 new documents. This is not to say that every added document is indeed useful as a reusable experience, but it indicates that daily work items were documented and shared.

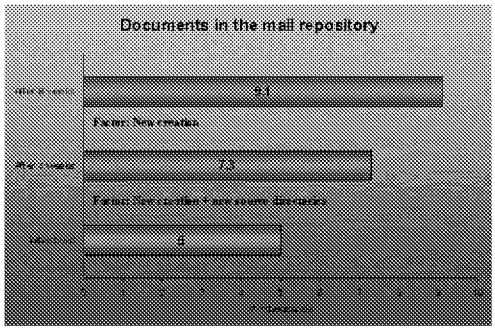


Figure 2 Growth of documents in the mail repository

In figure 3 we show the use of the repository over a seven week period. This figure shows that, in the early stages, people became more and more aware of the repository and more people tested the repository with their personal information needs (peak in the third bar). After that the usage frequency was lower, more stable and continuous,

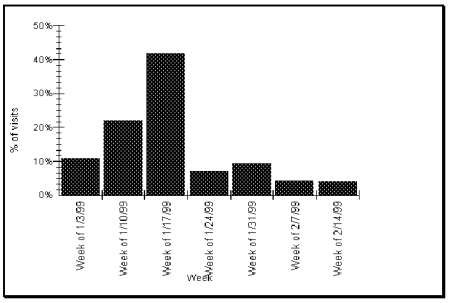


Figure 3. Usage of the Search Repository per week

indicating possible acceptance. In this figure, a visit is defined as a series of consecutive requests from a user to an Internet site and a request is a successful connection to an Internet site, i.e. retrieving contents. The graph shows the distribution of the number of different users visiting the web site as a percent of total visits over the period.

Figure 4 shows the average number of queries entered per visit per week. The number is low at the beginning. People were testing the repository with an average of one query, presumably to see the behavior and the functionality. Later the people seem to search more seriously for information. Further information provided to the department about the intent and use of the search repository probably caused the high increase at the end. The

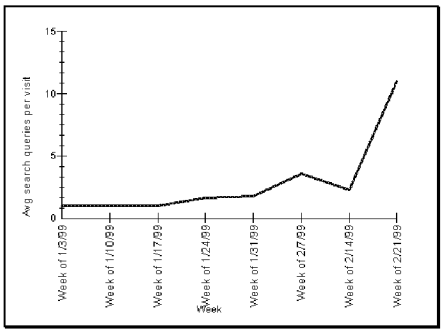


Figure 4. Average search queries per visit

combination of figures 3 and 4 is interesting. It shows that number of visits seems to be stabilizing but that the number of queries per visit is increasing. This demonstrates a relatively efficient usage pattern.

The more popular search queries entered during the last 8 weeks of the project were basically a binary classification of technology issues and process issues. The technology issues include ActiveX and XML. The process issues were classified as "process" in general and "estimation process". The data indicates a large diversity of information needs in the organizational environment. The repository was able to give back possibly useful documents for most queries. However we do not have any information to indicate whether the returned documents were useful or not.

Some queries did not return documents since the repository contained insufficient documents, for example new technologies like the XML language. As with the usage report, we need to be careful with the query data because it is only initial data from a short period of use.

The reports, although only initial, provide some preliminary indications.

- The acceptance, i.e. usage of the repository was promising.
- The usage frequency indicates a degree of integration into the daily information search activities.
- The information which was needed in the department covers a very wide range of areas.
- Process and new technology information seems to be of special interest.
- Informing people about the presence and the usefulness of the concept is important.

These were the initial conclusions from the limited data available. Surveying the repository users then extended these.

Survey about usage benefits

To get direct user feedback we decided to conduct an informal survey of staff impressions while using the search engine, ideas the users had, and the benefits realized through being able to search the local environment documents. Overall the acceptance and judgement of the product was good. The feedback ranged from ideas for extension, descriptions of how people used the repository, to first impressions. The following points capture the most common critical aspects, benefits, and extension ideas gained from the survey.

We found that the people who had been working in the department for a long time knew where information could be found without using the repository (e.g. document templates or whom to ask to get information). The opportunity for this will reduce as the department grows. We would then predict that the repository could play a stronger role in information transfer.

The benefits that were noted included comment that the search web site is a good address for new employees who are not familiar with the work environment. People also reported that they found documents and information that had been lost. The average time saved through this was estimated to be in a range of 1 to 4 hours. The search engine also reportedly breaks down information barriers between projects and environments (sharing experience & knowledge). It was seen as a good thing to first search for local information and experience before proceeding.

6. Conclusions

At the end of our implementation of cycle #1 we assessed which of our initial expectations for the defined approach were met. Earlier we described our expectations, which we now examine. The time our experience management environment (EME) was usable was 8 weeks and hence the underlying data has to be viewed carefully and further trends have to be monitored to prove the findings.

Our experience is that we generally achieved the technical objectives. In this respect the project was successful. We are relatively confident that the experience management environment could help support improvement in this environment. The data that was available at this time was too preliminary to justify strong conclusions about usage of the experience base. The usage patterns indicated a trend towards consistent use and integration into the work cycle. The project proved the viability of the bottom-up approach selected in this organization. Whether this will apply in other organizations clearly depends on many factors. We have outlined what we believe these factors to be. They range from broad organizational and cultural characteristics to technology characteristics. The most important evidence, we believe, is the clear establishment of a substantial experience base in an organizational setting in a short time period, which showed indications of successful deployment.

So what went wrong? Surprisingly, given the positive comments by the users, the system was decommissioned shortly after the completion of the project. A major contributor to this was the lack of ongoing management commitment to the project. While a senior manager was the initial champion of the project, its implementation was assigned to a busy project leader. In retrospect greater emphasis should have been placed on ensuring that the project champion maintained a more visible presence with respect to the experience factory project. A second issue was the lack of identification of clear goals and payback criteria for the project. It appears that, although technology can support this type of experience base development, a top down GQM-based methodology has the characteristics that are more likely to ensure longer-term success. The third observation was that the close physical proximity of the development teams and the relatively small number of personnel worked against the need for a more formal repository-based experience factory. The metrics success factors documented by Jeffery and Berry in [9] might provide an indicator of factors relevant to EF success as well. For example they list senior management commitment, realistic assessment of payback, clear responsibilities, determination of required granularity among many others. The issue of physical proximity has been observed by the authors in the context of electronic conferencing as a major implementation issue.

7. References

[1] Birk A. and Tutz C., "Knowledge Management of Software Engineering Lessons Learned", Fraunhofer IESE-Report No. 002.98/E, July, 1998

[2] Basili V., Caldiera G. and Rombach D., "The Experience Factory", in Marciniak J.J. ed., *Encyclopedia of Software Engineering*, John Wiley & Sons, 1994.

[3] PERFECT homepage, ESPRIT III project sponsored by the CEC, http://www.iese.fhg.de/Services/Projects/Public-Projects/Perfect/perfect.html

[4] Basili V., Caldiera G., McGarry F., Pajerski R., Page G. and Waligora S., "The Software Engineering Laboratory – An Operational Software Experience Factory", *Proc. of the International Conference on Software Engineering*, Melb, May, 1992, pp.370-381.

[5] Houdek F., Schneider K. and Wieser E., "Establishing Experience Factories at Daimler Benz: An Experience Report, *IEEE*, 1998, pp. 443-447.

[6] Basilli V. and Caldiera G., "Improve Software Quality by reusing Knowledge and Experience", *Sloan Management Review*, 37(1), 1995, pp. 55-64

[7] Basili V. and McGarry F. "The Experience Factory: How to Build and Run One", 17th International Conference on Software Engineering, Seattle USA, 1995

[8] Feldmann R. and Vorwieger S., "The web-based Interface to the SFB 501 Experience Base", SFB 501 Bericht 01/91, University of Kaiserslautern, 1991.

[9] Jeffery, R. and Berry, M. "A Framework for Evaluation and Prediction of Metrics Program Success", in Applying Software Metrics, edited by Paul Oman and Shari Lawrence Pfleeger, IEEE Computer Society, 1997, pp. 266-277.

Lessons Learned from the Failure of an Experience Base Initiative Using a Bottom-up Development Paradigm

Arne Koennecker
Uni. of Kaiserslautern/Fraunhofer IESE
Ross Jeffery & Graham Low
University of New South Wales

The Issues

1. A large electronic repository of lessons learned and project information.

2. The information is not easily accessed.

3. Question - Can we provide experience successfully without explicit packaging?

The Organizational Context

1. Well-documented processes
2. Existing measurement program
3. Staff committed to improvement and "allowed" to experiment

The Organizational Solution

Microsoft Site Server 3.0

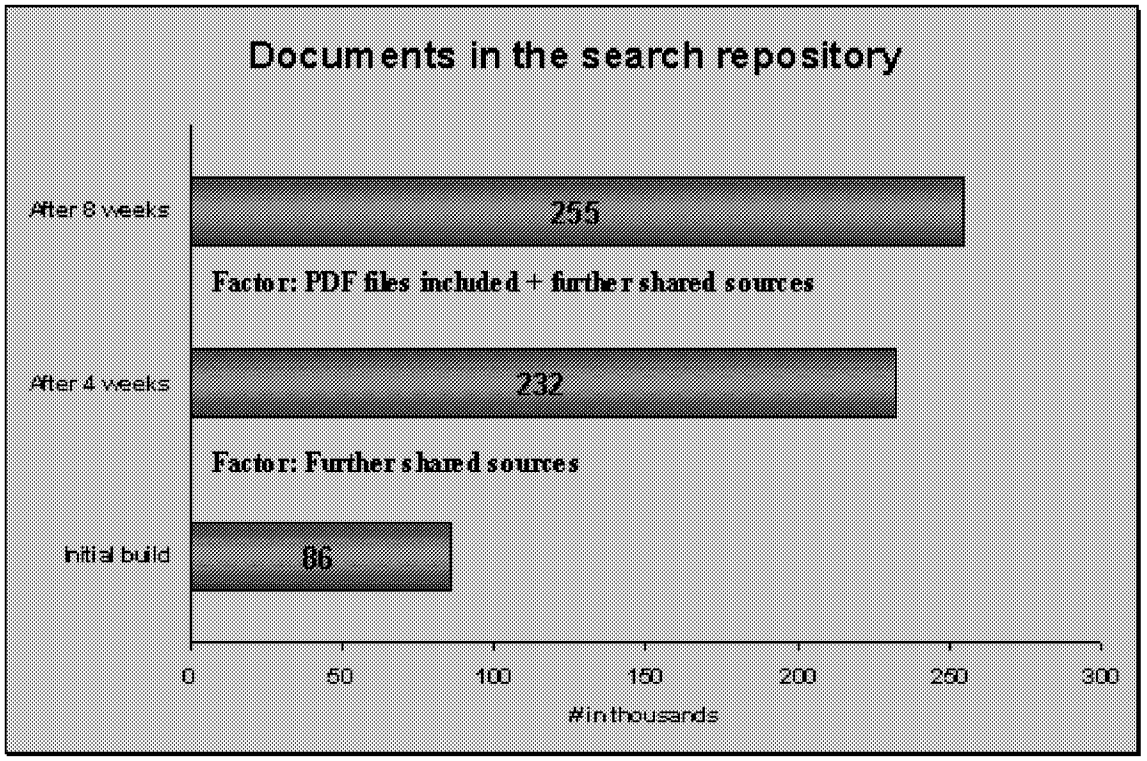
Network environment (Intranet) - Microsoft Windows NT .

Clients - Windows 98, Windows NT, Windows NT Server, Unix

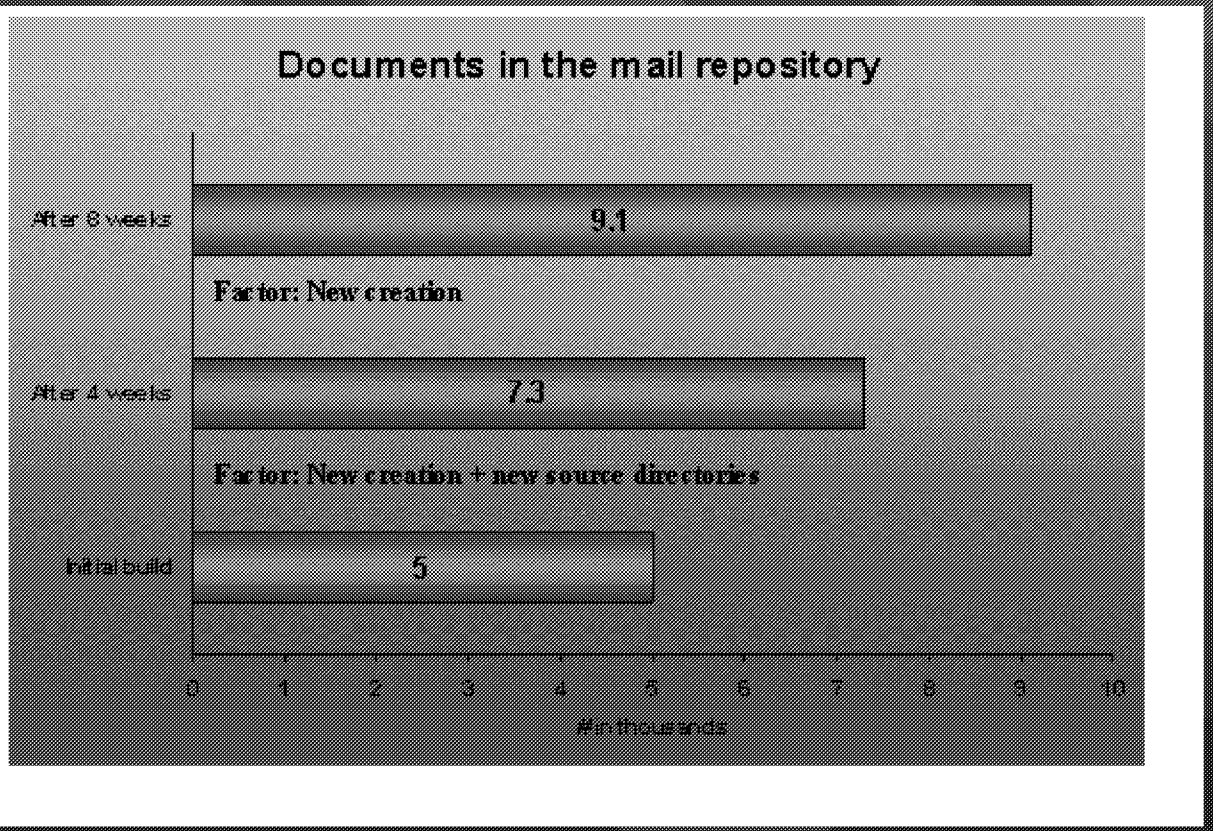
The Potential Experience Base

Document source	File types	Information type
Mail Exchange Server	Mail format (exch)	Folders for past and current project information, technology discussions, reuse items, etc.
Project and department web server	HTML, Microsoft Office documents (DOC, XLS, PPT), Adobe Acrobat PDF, database files (SQL, Access)	Project documents ranging from code to process descriptions, general department information like administration tasks
Local workstations	Microsoft Office documents (DOC, XLS, PPT), Adobe Acrobat PDF, HTML, plain text (TXT)	Documents gathered for own information purpose, document drafts

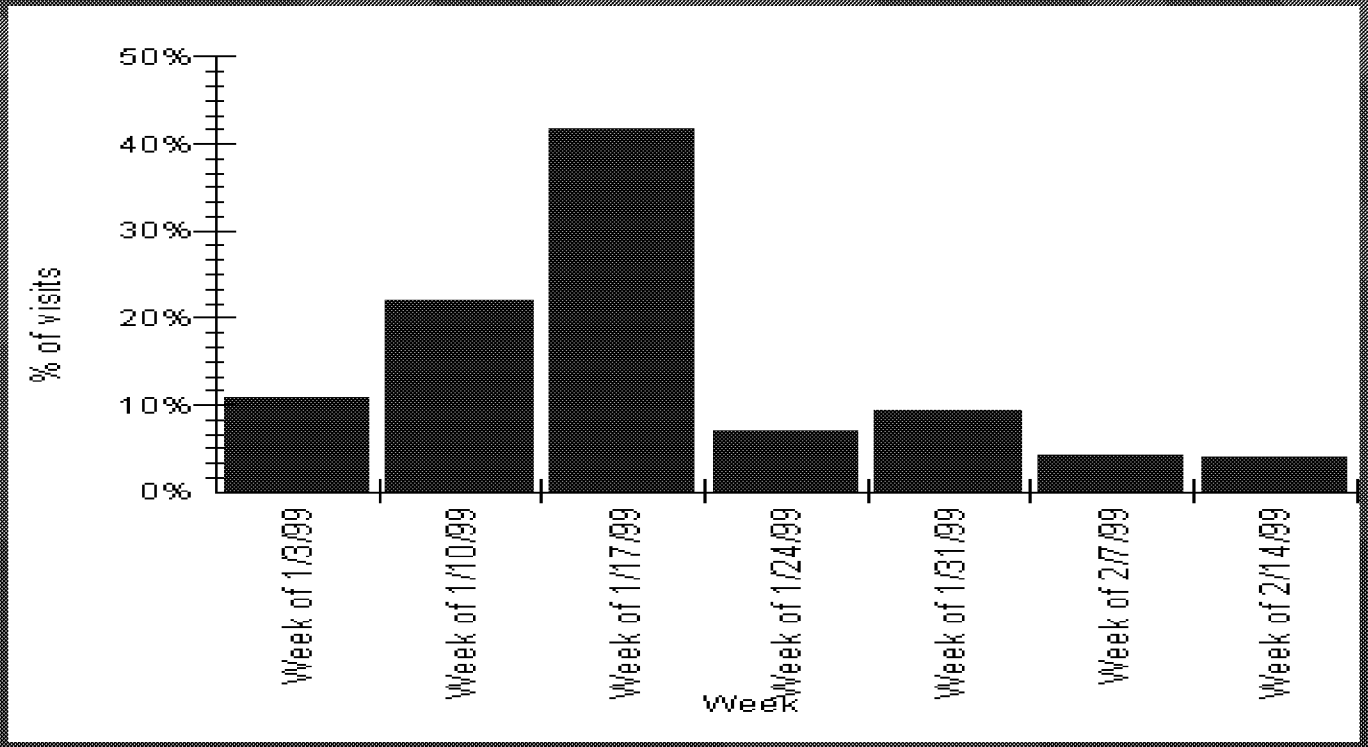
Analysis of Documents



Documents in the Mail Repository



Site Visit Pattern



Queries per Visit

Preliminary Indications.

- The acceptance and usage of the repository was promising.
- The usage frequency indicated a degree of integration into the daily information search activities.
- The information which was needed in the department covered a very wide range of areas.
- Process and new technology information seemed to be of special interest.
- Informing people about the presence and the usefulness of the concept was important.

Survey about usage benefits

•People who had been working in the department for a long time knew where information could be found without using the repository.

•People also reported that they found documents and information that had been lost. The average time saved through this was estimated to be in a range of 1 to 4 hours.

•The search engine breaks down information barriers between projects and environments.

Conclusions (1)

The environment was usable for only 8 weeks and hence the data has to be considered very carefully.

Our experience is that we achieved the technical objectives. In this respect the project was successful.

The usage patterns indicated a trend towards consistent use and integration into the work cycle.

Conclusions (2)

Surprisingly, given the positive comments by the users, the system was decommissioned shortly after the completion of the project.

While a senior manager was the initial champion of the project, its implementation was assigned to a busy project leader.

1. Greater emphasis should have been placed on ensuring that the project champion maintained a more visible presence on the project.

Conclusions (3)

2. There was a lack of identification of clear goals and payback criteria for the project.

Although technology can support this type of experience base, a top down GQM-based methodology has the characteristics that are more likely to ensure longer-term success. A business focus might have maintained the initiative.

3. The close physical proximity of the development teams and the relatively small number of personnel worked against the need for a more formal repository-based experience factory.

An Experience Management System for a Software Consulting Organization

Carolyn Seaman[†*] cseaman@umbc.edu
Manoel Mendonça[§*] manoel@cs.umd.edu
Victor Basili[§*] basili@fc-md.umd.edu
Yong-Mi Kim[‡] yong-mi.kim@q-labs.com

[§]*University of Maryland at College Park*

[*]*Fraunhofer Center for Experimental Software Engineering*

[†]*University of Maryland at Baltimore County*

[‡]*Q-Labs, Inc.*

1 Introduction

Software is a major expense for most organizations and is on the critical path to almost all organizational activities. Individual software development organizations in general strive to develop higher quality systems at a lower cost for both their internal and external customers. Yet the processes used to develop such software are still very primitive in the way that experience is incorporated. Learning is often from scratch, and each new development team has to relearn the mistakes of its predecessors. Reuse of an organization's own products, processes, and experience is becoming more accepted as a feasible solution to this problem. But implementation of the idea, in most cases, has not gone beyond reuse of small-scale code components in very specific, well-defined, situations. True learning within a software development organization requires that organizational experiences, both technological and social, be analyzed and synthesized so that members of the organization can learn from them and apply them to new problems.

Suppose, for example, that a member of a software development group is considering the use of a particular software engineering technology on a forthcoming project. This member has heard that this technology has been used successfully in other projects in some other part of the organization, but cannot easily find out where or by whom. He or she would like very much to learn from the experiences of those previous projects, first to help make the decision to use the technology or not, then to help implement the technology in the current project. It would be helpful, obviously, to avoid the inevitable mistakes that are made the first time a new technology is tried. Also, it would be useful to see the costs of using that technology (e.g. the costs of new tools or training) in order to help estimate those costs for the current project. Without the organizational infrastructure to support access to previous experience from within the organization, this type of information would be very difficult, if not impossible, for the development team member to get.

This paper describes a system for supporting experience management in a multinational software improvement consultancy called Q-Labs. This Experience Management System (EMS) is based on the Experience Factory concept [1] proposed by Basili. This paper focuses on describing the design principles behind EMS and reports the results of an evaluation of its interface.

2 The Experience Factory

Basili proposed the Experience Factory as an organizational infrastructure to produce, store, and reuse experiences gained in a software development organization [1,2,3]. The Experience Factory idea organizes a software development enterprise into two distinct organizations, each specializing in its own primary goals. The Project Organization focuses on delivering the software product and the Experience Factory focuses on learning from experience and improving software development practice in the organization. Although the roles of the Project Organization and the Experience Factory are separate, they interact to support each other's objectives. As illustrated in Figure 1, the feedback between the two parts of the organization flows along well-defined channels for specific purposes. Also, the Experience Factory

supports the meta process defined by Basili's Quality Improvement Paradigm (QIP) [6]. As shown in Figure 1, for each new project: the problem at hand is characterized (1), goals are set (2), a suitable process is chosen (3), the process is executed and measured (4), outputs are analyzed (5), and lessons and products are packaged and stored in the experience base for future reuse (6).

Experience Factories recognize that improving software processes and products requires: (1) continual accumulation of evaluated and synthesized experiences in *experience packages*; (2) storage of the experience packages in an integrated *experience base* accessible by different parts of the organization; and (3) creation of *perspectives* by which different parts of the organization can look at the same experience base in different ways. Some examples of experience packages might be the results of a study investigating competing design techniques, a software library that provides some general functionality, or a set of data on the effort expended on several similar projects.

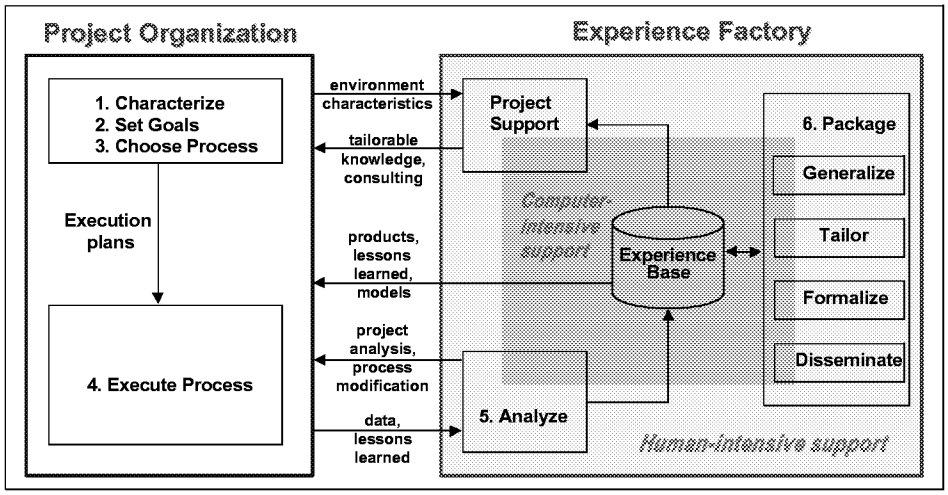


Figure 1. Experience Factory structure

The Experience Factory concept has been implemented in a number of software development organizations that have addressed the above questions in various ways (e.g. [4,5,9]). The Software Engineering Laboratory (SEL) [4] is an example of an Experience Factory. The SEL Quality Improvement Paradigm provides a practical method for facilitating product-based process improvement within a particular organization. Because it directly ties process improvement to the products produced, it allows an organization to optimize its process for the type of work that it does. Using this approach, the SEL has reduced development costs by 60%, decreased error rates by 85%, and reduced cycle time by 20% over the past 10 years. Establishing an Experience Factory, however, is a long-term endeavour requiring a great deal of commitment on the part of both management and development staff. Implementing an Experience Factory involves substantial up-front costs. It requires instilling a new philosophy of learning into an organization, establishing an organizational structure and processes for the Experience Factory to collect, package and share experiences. Once in place, it will also require substantial ongoing effort and commitment to maintain itself as an effective agent for continuous software process improvement.

We believe that emerging computing technologies – such as distributed systems, visual query interfaces, and intranets – offer great potential to support the establishment and maintenance of Experience Factories in organizations. This paper reports preliminary results and experiments from a research project aimed at implementing a system for supporting an Experience Factory within an industrial setting.

3 The Principles Behind the Experience Management System

We have found it useful to discuss the problem of software experience capture and reuse, and our approach to addressing it, in terms of the 3-layer conceptual view shown in Figure 2. This view shows three aspects of the problem, all of which need to be addressed before a complete solution can be implemented. At the lowest level, there are issues of how experience should be electronically stored in a repository and made accessible across geographical boundaries. The middle level deals with user interface issues, including how experiences are best presented to a user and how the user interacts with the automated system to

manipulate, search, and retrieve experience. At the top level, the organizational issues of how experience reuse will fit into the work of the organization, how the experience base will be updated and maintained, and how experiences will be analyzed and synthesized over time, are addressed. The bottom two levels of Figure 2 define the computer-intensive support pictured in Figure 1. The top level of Figure 2 defines the interface between the human-intensive and the computer-intensive areas described in Figure 1.

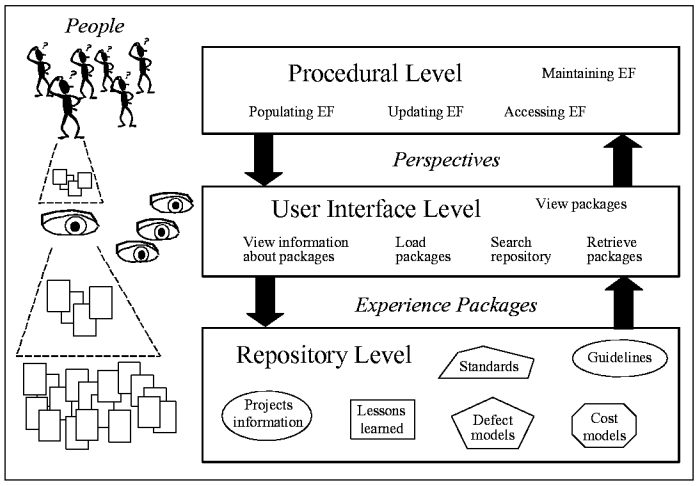


Figure 2. The three levels of an Experience Management System

Allied with this conceptual view, we have defined a set of requirements aimed at making the EMS reliable, easy to use, and flexible enough to support the Experience Factory concept.

R1. The system shall support geographically distributed organizations allowing them to share and manage experience packages remotely.

R2. The repository shall be robust, reliable, and portable to standard computer platforms.

R3. The user interface level shall be as platform independent as possible.

R4. The data model shall be simple but powerful enough to model diverse classes of "experience packages." The system will adapt to the current practices, processes, and products of different organizations, and not vice-versa.

R5. The system shall be easy to learn and self explanatory. The user interface shall be easy to use and the stored information shall be easy to search and retrieve.

This conceptual view, along with the requirements, form the basis of several ongoing efforts to implement experience management systems in a variety of settings. The first of these efforts, the Q-Labs EMS, is described in this paper. Lessons learned from our work with Q-Labs will be fed into other EMS efforts in the future.

4 The Q-Labs EMS

The Experimental Software Engineering Group (ESEG) at the University of Maryland and Q-Labs, Inc., have been working together for nearly three years on a project aimed at building the infrastructure to support a true Experience Factory within Q-Labs, resulting in an Experience Management System (from here on called the "Q-Labs EMS"). Q-Labs is a multi-national software engineering consulting firm that specializes in helping its clients improve their software engineering practices by implementing state-of-the-art technologies in their software development organizations. Q-Labs has helped many of its clients implement some of the principles of the Experience Factory. Q-Labs' objectives for this project have been to provide a "virtual office" for the organization, which is spread across two continents, and to allow each Q-Labs consultant to benefit from the experience of every other Q-Labs consultant.

4.1 System Architecture

In order to fulfill the first requirement presented in section 3, to support geographically distributed organizations, the Q-Labs EMS is a client-server system. The clients enforce the policies defined at the procedural level and implement the system front-end applications defined by the user interface level (here referring to the levels in Figure 2). The server implements the system repository. The architecture of the system is shown in Figure 3. It follows a three-tier model. At the top level, we have the EMS Manager and EMS Visual Query Interface (VQI) applications. They work as client applications sending requests to a "middle tier" of services. This "EMS Server" receives messages from the client applications and translates them into low-level SQL (Standard Query Language) calls to an "EMS Repository."

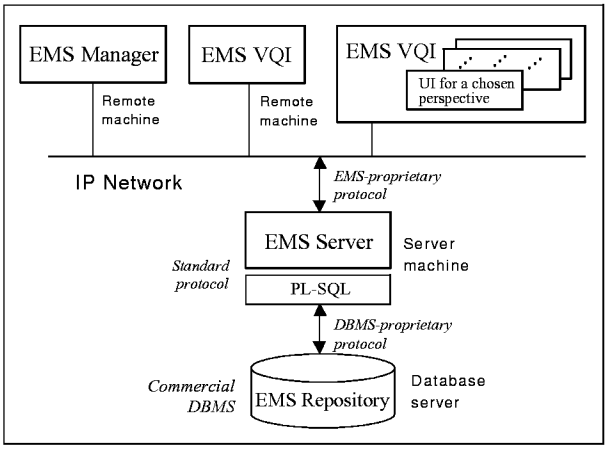


Figure 3. Q-Labs EMS architecture

In order to fulfill the second requirement, repository robustness and portability, the EMS Repository uses standard database technology. It stores all the information necessary for the EMS operation in a relational database managed by a commercial DBMS (Data Base Management System.) The link between the server and the repository is done through standard embedded SQL (PL-SQL.) This makes the repository portable to standard commercial DBMS, and virtually portable to any platform.

In order to fulfill the third requirement, a platform independent user interface, the client applications are implemented in Java™. This makes them portable to any platform that has a Java virtual machine. To date we have tested the client applications – EMS Manager and EMS VQI – on Unix, Windows (NT and 98), and Macintosh platforms.

4.2 Data Model

An early, crucial task in this project has been identifying the pieces of information that should be packaged as experience. However, each organization has different needs and experiences. In order to fulfill the fourth requirement, data model simplicity and flexibility, we introduce the concept of a **perspective**. A *perspective* defines a class of packages much like an *object class* in an object-oriented system. A perspective is defined by three parts: a classification part, a relationship part, and a body part.

The classification part, called the perspective **taxonomy**, defines a classification model for the packages instantiated from that perspective. The perspectives' taxonomies describe the contents of an experience base in an organization's own terminology, thus guiding users intuitively towards the experiences of interest to them. A taxonomy is composed of **attributes** with well-defined naming and typing. The attributes effectively define the facets that can to be filled by an experience packager to characterize a package instantiated in a given perspective.

The relationship part, called the perspective's **links**, defines the relationship between the packages instantiated in this perspective and other packages in the experience base. Like attributes, links have a name and type associated with them.

The perspective **body** defines the **elements** that compose the experience packages instantiated from this perspective. Like attributes and links, elements have names and types. The type is usually a file or a list of files. Those files are internally stored in the experience base as large objects when a package is instantiated from a perspective.

4.3 Visual Query Interface

In order to fulfill the fifth requirement, a search and retrieval interface that is easy to learn and self-explanatory, we adopted a visual query interface (VQI) concept. As proposed by Shneiderman [13], visual query interfaces let users "fly through" stored information by adjusting widgets and viewing animated results in the computer screen. In EMS, they allow easy interactive querying of the repository based on various attributes of the experience packages. Built in to the interface is the set of attributes defined for the perspective currently being viewed. Figure 5 shows the user interface for the Q-Labs EMS. Upon login a user will have a set of perspectives from which he/she can look at stored experience packages. A user will fire a VQI by selecting one of those perspectives. The VQI will display the packages that are associated with this perspective together with the attributes and query devices (slider bars, check boxes, etc.) used to search and browse those packages. The widgets used on the interface are defined on the fly based on the data types and number of different values associated with each attribute.

Using the VQI, the user interactively searches the experience packages associated with a certain perspective by manipulating the widgets on the right and observing the number of selected packages on the two-dimensional chart. Once a small subset of packages is selected using the VQI query devices, the user can quickly examine specific packages by clicking on them. This will fire a Web Page with a complete description of the selected package, including its links and elements. If the selected package corresponds to the user's expectations, he/she can click on the desired elements to retrieve the package's files.

The VQI has two features that we believe are fundamental to EMS. First, its search is interactive and controlled by the user. This allow the user to easily control the number of matches by widening or narrowing the search scope with a few mouse clicks. This is a clear advantage over keyword-based search – such as those executed by Worldwide Web search engines. We hypothesize that this significantly helps users to find packages that are useful to them even when an exact match is not available. The second key feature of this type of interface is that it allows people to visualize the amount of stored experience and the classification schema used by the organization. We believe that this significantly helps new users to get used to EMS and is also an important learning medium for new team members.

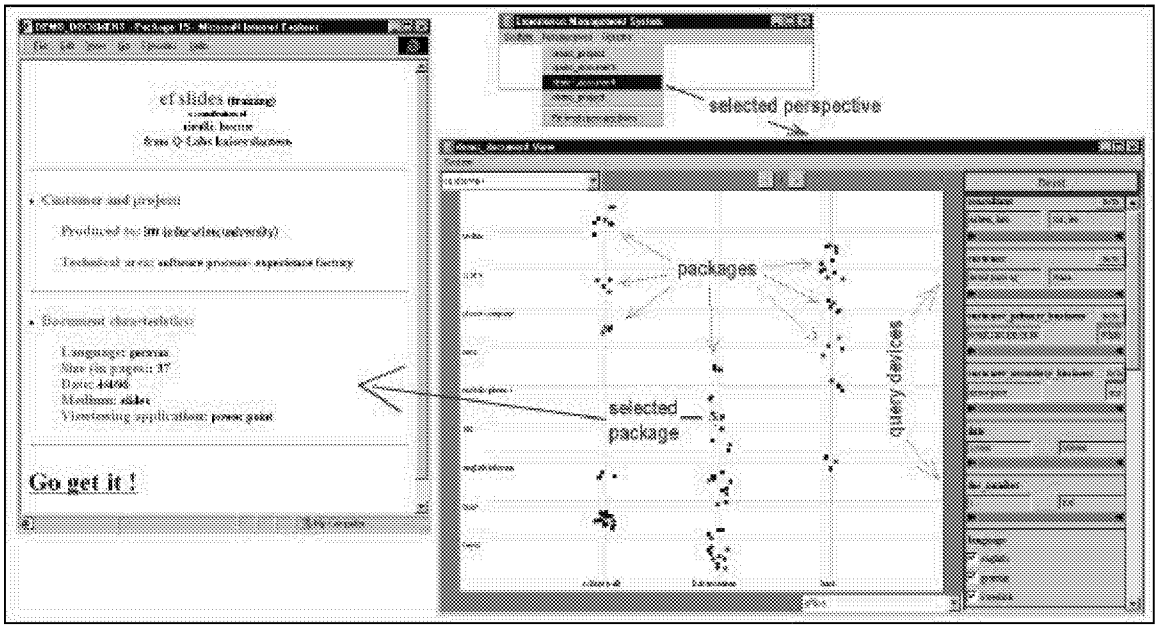


Figure 4. Q-Labs Visual Query Interface (VQI)

The user interface also has functionality to allow users to submit new experience packages to the experience base. This functionality uses the attributes, links, and elements associated with the perspectives to produce the forms that a user must complete to describe new packages.

5 Interface prototype evaluation

The first of several planned empirical studies to evaluate the Q-Labs EMS prototypes was an evaluation of the interface. This initial prototype consisted of the VQI (pictured in Figure 5), a simple data entry interface used to submit experience packages (just a form with each field corresponding to one of the defined attributes for a given perspective), and a small repository populated with a collection of real Q-Labs documents and project descriptions. Two perspectives were also provided with this prototype. The *documents* perspective used attributes of documents (e.g. author, date, title, etc.) as the search mechanisms, while the *projects* perspective used attributes of projects (e.g. project lead, customer, start date, finish date, total effort, etc.). Some attributes were common to both perspectives (e.g. technical area). The evaluation was carried out at this point in the project (before having a full working system) because it was essential to get user feedback on the basic paradigms we had chosen before we proceeded further.

5.1 Study design

The interface evaluation study was based on qualitative methods [10]. The importance of such methods in validating software engineering technology is discussed by Seaman in [12]. The goals of the interface evaluation study were:

1. To evaluate the current set of attributes (in both the "projects" and "documents" taxonomies) in terms of completeness, usefulness, and clarity.
2. To evaluate the visual query search and retrieval interface in terms of usefulness, usability, and appropriateness.
3. To evaluate the data entry interface in terms of feasibility, usability, and impact on working procedures.

These goals were refined into a set of questions that guided the design of the study. To answer these questions, two types of data were collected. The first data source consisted of detailed notes concerning how the subjects used the prototype and the comments they made while using it. The second data source came from a set of interviews that were conducted at the end of each evaluation session. The questions asked during the interviews are shown below in Figure 5.

1. What did you like most about the search and retrieval interface?
2. Was there anything really annoying about using it?
3. Was it easy to move around and do things with the mouse and keyboard?
4. Is there any information that would be useful to include in the interface that isn't there?
5. Are there any attributes that are not clear in their meaning?
6. What attributes did you use most in searches?
7. Did you feel that you were able to find what you were looking for using the interface?
8. How satisfied are you that tasks can be completed with the minimal number of steps?
9. How could the interface be improved?
10. Do you think you would use this tool, once the database was populated, in your everyday work? Does it support the way you normally work?
11. What did you like most about the data entry interface?
12. Was there anything really annoying about using it?
13. Were there any parts of the data entry interface where it wasn't clear what information you should enter?
14. How was using this interface different from the usual procedure for recording this type of information? Do you think, in general, that this would save you time or not?
15. How could the interface be improved?
16. Do the different parts of the system have consistent appearance and work in similar ways?
17. How satisfied are you with the system appearance in terms of color, layout, and graphics usage?

Figure 5. Evaluation Interview Questions

Interface evaluation sessions were held with five different Q-Labs consultants from three different offices in May and June of 1999. In each session, the subject was given a short hands-on training, then given a set of exercises that represented common Q-Labs work scenarios. The exercises were taken from the set of use cases we had collected as part of the initial requirements gathering activity for EMS. The subjects were asked to choose some of the exercises and then to use the Q-Labs EMS prototype to gain information

relevant to the scenario described in each exercise. They were also asked to verbalize their thoughts and motivations while working through the exercises. This technique, called a "think aloud" protocol [8], is often used in usability studies (and occasionally in other software engineering studies [14]) to capture a subject's immediate impressions, thought processes, and motivations while performing a task. The subjects could and did ask questions of the researcher conducting the session. After several exercises had been completed, a short interview was conducted, using the questions presented above as an interview guide. All the sessions were audiotaped and observed by at least one researcher. Each session lasted about 1.5 to 2 hours. Although the tapes were not transcribed verbatim, they were used to write very detailed notes after the fact.

The notes written from the tapes served as the major data source for the analysis part of the study. The analysis method used was the constant comparison method [7,10]. This method begins with coding the field notes by attaching codes, or labels, to pieces of text that are relevant to a particular theme or idea that is of interest in the study. Then passages of text are grouped into patterns according to the codes and subcodes they've been assigned. These groupings are examined for underlying themes and explanations of phenomena. The next step is the writing of a field memo that articulates a proposition (a preliminary hypothesis to be considered) or an observation synthesized from the coded data. In this case, the field memo written as part of this process became the results of the study, which are reported in the next section.

5.2 Results

The subjects generally liked the basic elements of the search and retrieval interface. In particular, they seemed to have no trouble mastering the search mechanism and liked how it was easy to negotiate the interface and see the distribution of packages among different attribute values. They also liked the immediate feedback in the graph part of the interface in response to changes made with the search mechanisms. Subjects were also able to glean useful information from the interface even when they couldn't find exactly what they were looking for. For example, one subject found a document that was not exactly what she wanted, but she saw the primary author's name and decided that would be a good contact, and so she felt she had found useful information.

The learning curve on the search and retrieval interface was fairly short. By the second or third exercise tried, all of the subjects were conducting their searches very rapidly and confidently. For some subjects, it was even quicker. Subjects generally narrowed their searches down to about 2-4 "hits" before looking at individual packages. This was seen as a "reasonable" number of packages to look through.

Several major annoyances surfaced during the evaluation. One was the use of slider bars. Several subjects had trouble figuring out the mechanics of using and interpreting them. Several subjects suggested using some form of checkboxes instead of the slider bars. Another annoyance had to do with the relationship between the two perspectives and the lack of linkage between them. After finding some relevant project in the projects perspective, subjects had to then start up the document perspective and start a search from scratch in order to find documents related to the project. A related problem was the confusion caused by some attributes and attribute values existing in one perspective but not the other.

As for the data entry interface, the data being collected was seen to be appropriate, but otherwise it left a lot to be desired. Subjects in general found the data entry interface unusable because they needed more guidance as to what attribute values to enter in the fields. Almost all of the subjects suggested pull-down menus or automatic fill-ins to decrease the amount of typing and increase consistency. In general, the subjects saw this interface as just a skeleton of what was needed.

All of this was valuable feedback that has been used in our plans for further development of the Q-Labs EMS. Although we knew that the interface we were evaluating was not ideal, we had not anticipated some of the specific problems that our subjects reported. For example, we had not considered the slider bar mechanism to be a problem, but our subjects definitely did. Also, although we knew the data entry interface needed some improvements (many of the suggestions from the subjects were already in our development plans), we had not considered it as completely unusable as our subjects did. On the other hand, the study validated some of our basic choices in the interface design, e.g. the VQI and the use of attributes and perspectives. Thus we can, with confidence, continue improvement of the interface without changing the underlying structure.

There were also some lessons learned about how the interface evaluation was conducted. Some problems came up related to the limited scope of the repository. Subjects were sometimes frustrated when there was nothing to be found for their search criteria. Subjects were also bothered by inconsistencies in the sample data. In particular, one subject found that there was a document in the documents perspective, that had a project name associated with it, but that project was not to be found in the projects perspective.

The interface evaluation, in general, proved to be a valuable and timely tool for getting feedback from the eventual users of the Q-Labs EMS. The effort involved was relatively small, although finding and scheduling subjects was difficult and caused some delays. Although much remains to be done before an operational system is delivered, the evaluation assured us that the Q-Labs EMS will eventually be acceptable to its intended users. In addition, the evaluation provided an opportunity to disseminate the aims of our project, and our work thus far, throughout Q-Labs.

6 Conclusions

We have described an ongoing project involving the Experimental Software Engineering Group (ESEG) at the University of Maryland and Q-Labs, Inc. that aims to provide a system (with both organizational and automated elements) to support software engineering experience capture and reuse. The current design of this system, called the Q-Labs EMS, is outlined, in particular its architecture and its user interface. Currently, an interface prototype exists and has been evaluated. This evaluation is described in detail. The results of the evaluation have assured us not only that the Q-Labs EMS will eventually be successfully deployed throughout Q-Labs, but will also serve as a testbed for our further investigation of software experience capture and reuse. However, much needs to be done before a working version of this system is in place. The prototype that has been evaluated encompassed only some of the automated features of the system. Much of the technical work remains, as well as the organizational part of the system. The latter includes designing, implementing, and evaluating new organizational procedures and deployment strategies to ensure the acceptance of EMS at Q-Labs.

7 References

[1] Basili, Victor R. Software Development: A Paradigm for the Future. *In Proc. of COMPSAC '89*, Orlando, Florida, pp. 471-485, September 1989.

[2] Basili, Victor R., and Gianluigi Caldiera, Improve Software Quality by Reusing Knowledge and Experience. *Sloan Management Review*, MIT Press, Volume 37, Number 1, Fall 1995.

[3] Basili, Victor R., Gianluigi Caldiera, and H. Dieter Rombach. The Experience Factory. In *Encyclopedia of Software Engineering*, New York: John Wiley & Sons, 1994. pp. 470-476.

[4] Basili, Victor R., Gianluigi Caldiera, Frank McGarry, Rose Pajerski, G. Page, and S. Waligora. The Software Engineering Laboratory - an Operational Software Experience Factory. *In Proceedings of the International Conference on Software Engineering*, May 1992, pp. 370-381.

[5] Basili, Victor, Michael K. Daskalantonakis, and Robert H. Yacobellis. Technology Transfer at Motorola. *IEEE Software*, March 1994, pp. 70-76.

[6] Basili, Victor, and H. Dieter Rombach. The TAME Project: Towards improvement-oriented software environments. *IEEE Transactions on Software Engineering*, 14(6):758-773, June 1988.

[7] B.G. Glaser and A.L. Strauss. *The Discovery of Grounded Theory: Strategies for Qualitative Research*. Aldine Publishing Company, 1967.

[8] J.T. Hackos and J.D. Redish. *User and Task Analysis for Interface Design*. New York:John Wiley and Sons, 1998, chapter 9, pp. 258-9.

[9] Houdek, F., K. Schneider, and E. Wieser (April 1998). Establishing Experience Factories at Daimler-Benz: An Experience Report. In *Proc. of 20th International Conference on Software Engineering*, Kyoto, Japan, pp. 443-447.

[10] M.B. Miles and A.M. Huberman. *Qualitative Data Analysis: An Expanded Sourcebook*, second edition, Thousand Oaks:Sage, 1994.

[11] Prieto-Diaz, R. Classifying of Reusable Modules. In T.J. Biggerstaff and A. Perlis, editors, *Software Reusability*, Volume I, ACM Press, 1990.

[12] Seaman, C.B. Qualitative Methods in Empirical Studies of Software Engineering. IEEE Transactions on Software Engineering, 25(4):557-572, July/August 1999.

[13] Shneiderman, Ben. Dynamic Queries for Visual Information Seeking. *IEEE Software*, Vol. 6, No. 11, November 1994, pp 70-77.
[14] A. von Mayrhauser, and A.M. Vans. Identification of dynamic comprehension processes during large scale maintenance" *IEEE Transactions on Software Engineering*, 22(6):424-437, June 1996.

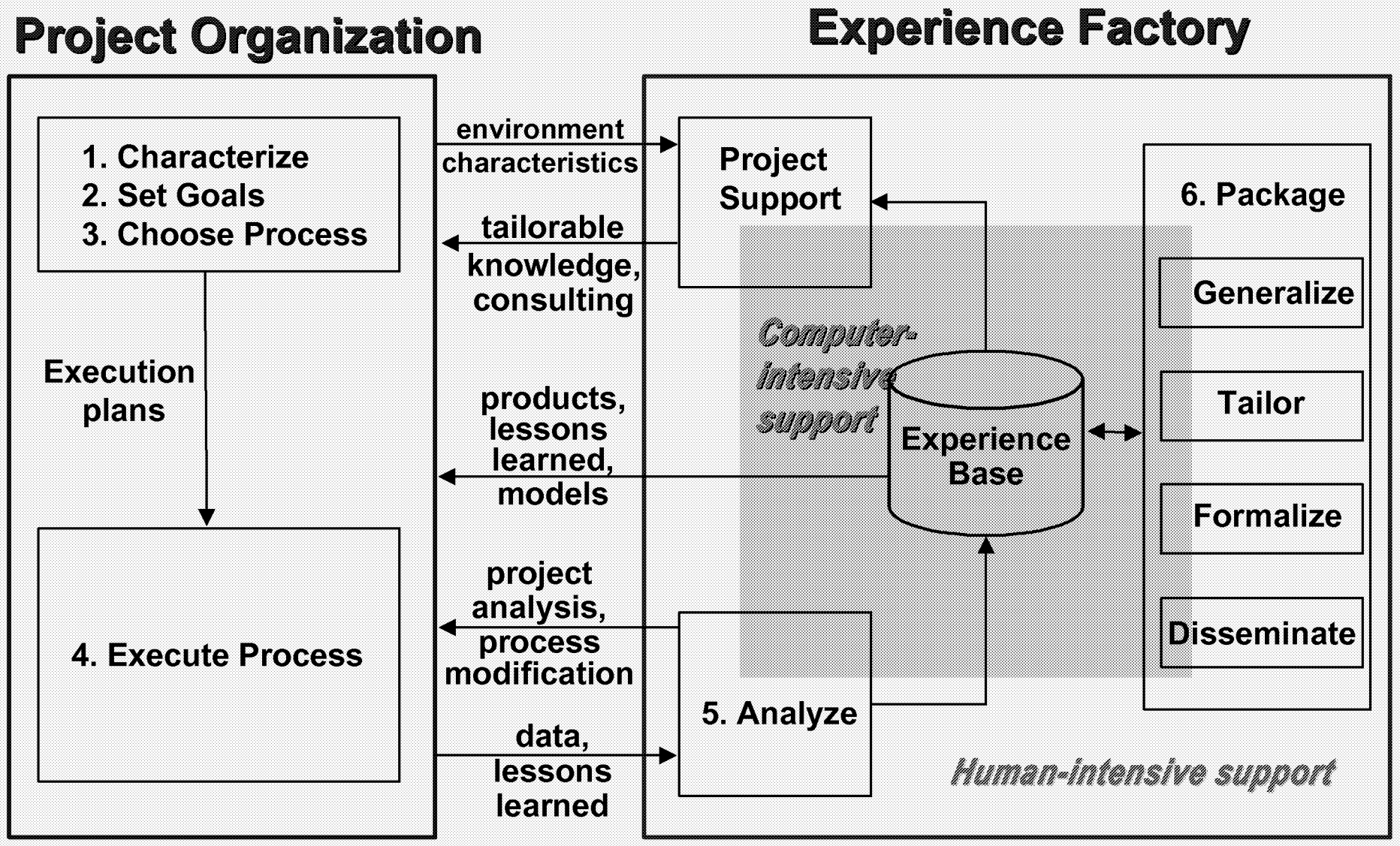
An Experience Management System for a Software Consulting Organization

Carolyn Seaman
Manoel Mendonça
Victor Basili
Yong-Mi Kim

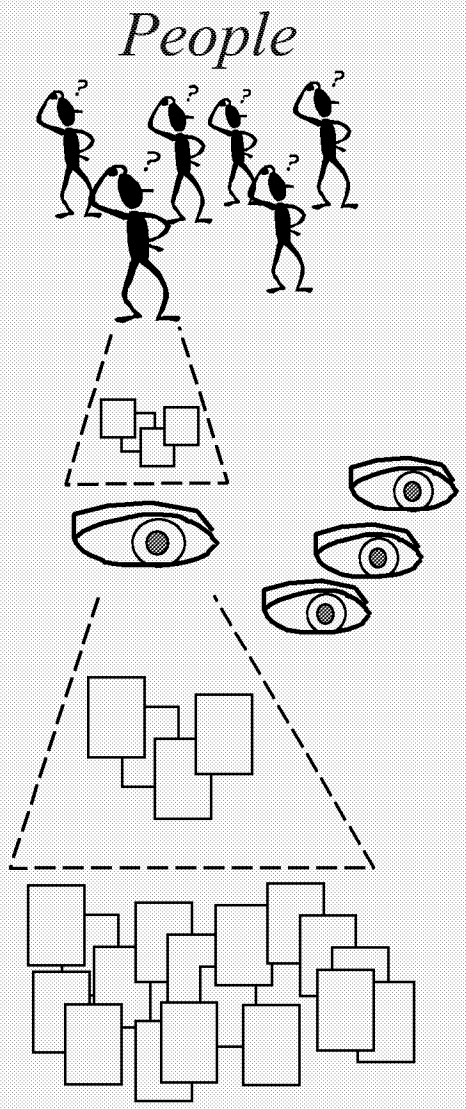
Organizational Affiliations:
Q-Labs, Inc.
University of Maryland College Park
University of Maryland Baltimore County
Fraunhofer Center - MD

Funding is provided by Q-Labs, Inc. and the Maryland Industrial Partners (MIPS) program

The Experience Factory



Experience Factory Support



Procedural Level

Populating EF | Updating EF | Accessing EF | Maintaining EF

Perspectives

User Interface Level

View information about packages | Load packages | Search repository | Retrieve packages | View packages

Experience Packages

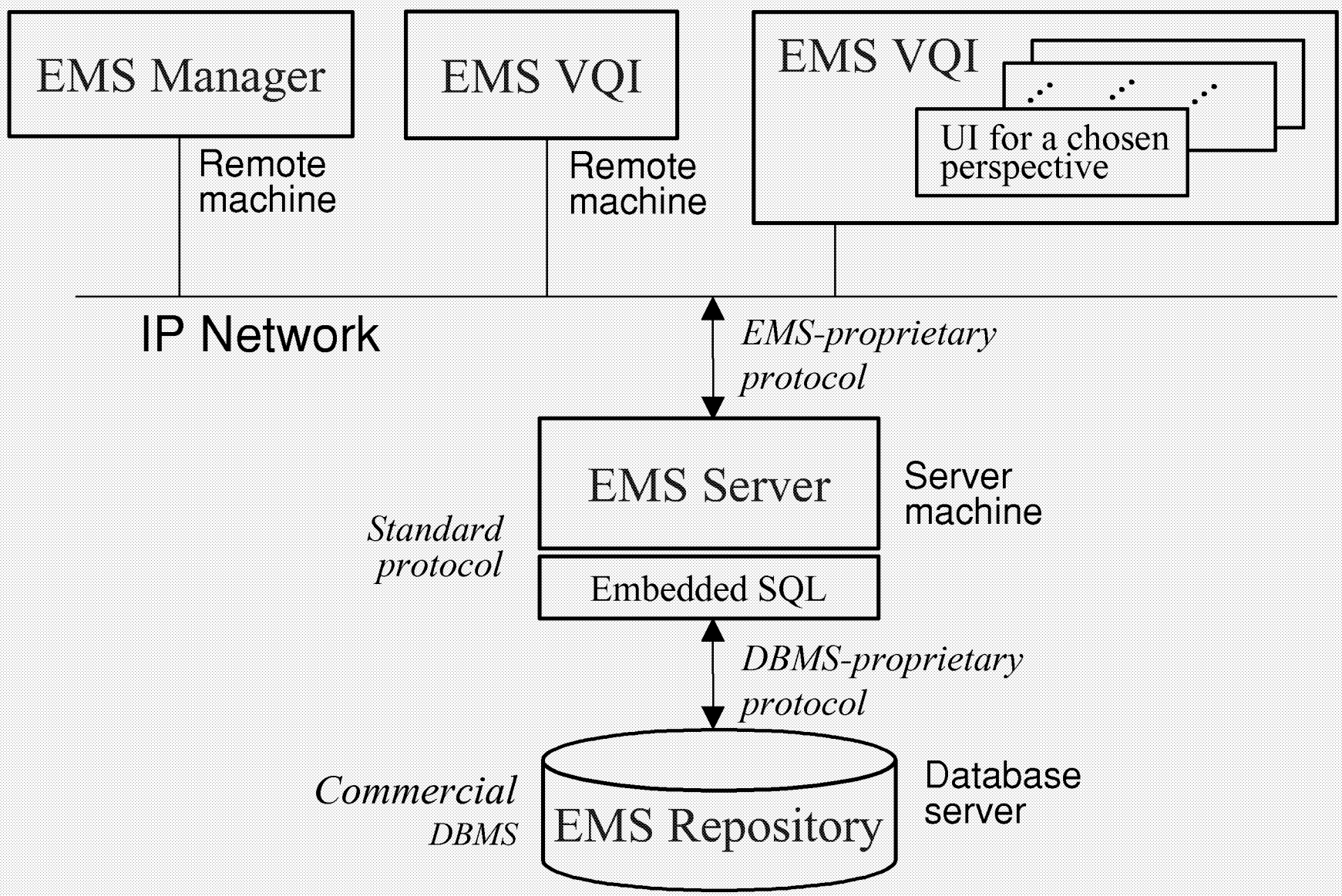
Repository Level

Projects information | Lessons learned | Defect models | Cost models | Standards | Guidelines

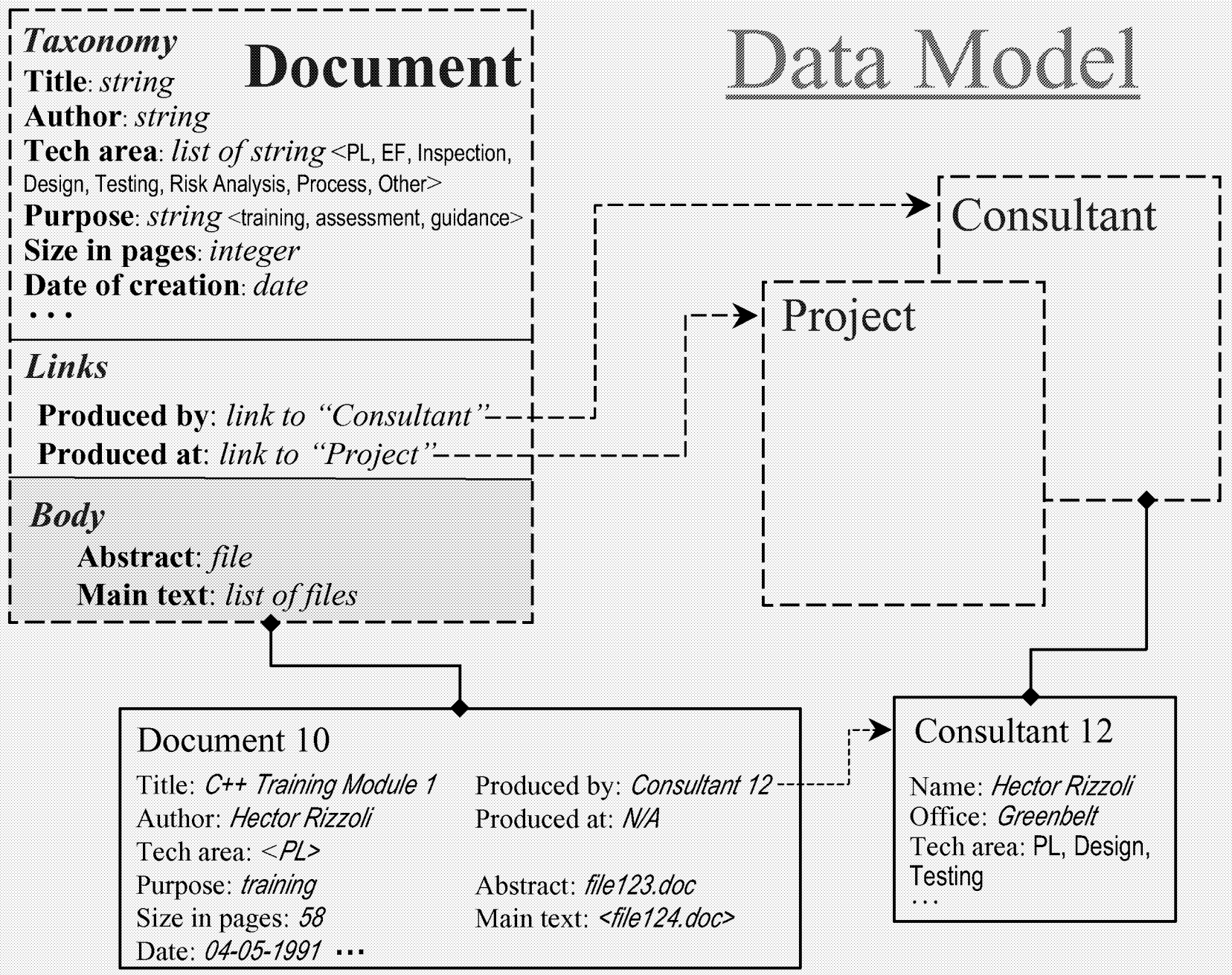
The Q-Labs EMS

- Q-Labs, Inc.
  - a multi-national software engineering consulting firm helping its clients implement state-of-the-art technologies
  - objectives: to allow each consultant to benefit from the experience of every other consultant
- EMS project
  - multi-faceted research effort in developing appropriate infrastructure for an Experience Factory
  - Q-Labs EMS is the first prototype EMS

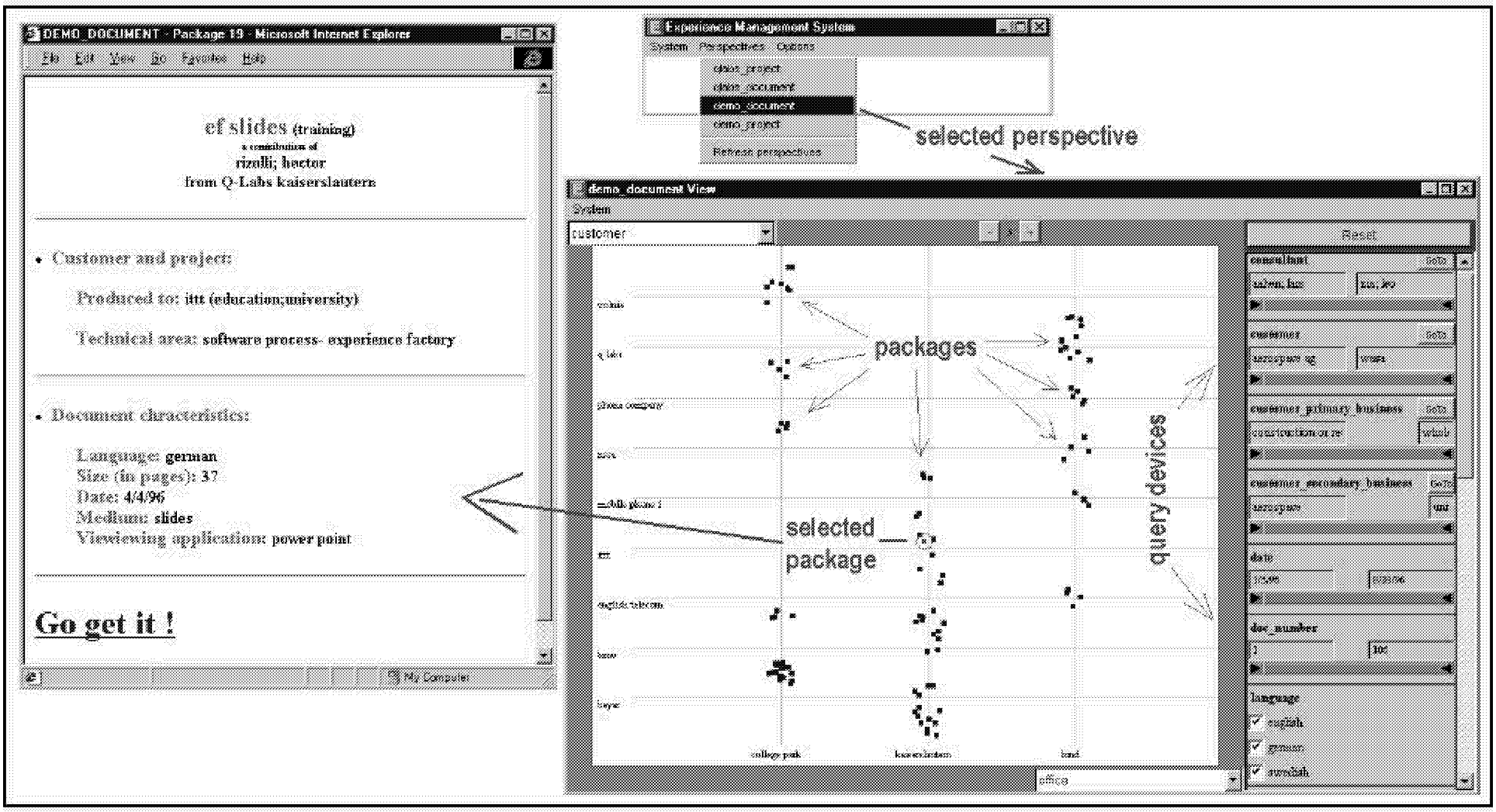
System Architecture



Data Model


Taxonomy
Document
Title: string
Author: string
Tech area: list of string <PL, EF, Inspection, Design, Testing, Risk Analysis, Process, Other>
Purpose: string <training, assessment, guidance>
Size in pages: integer
Date of creation: date
• • •
Links
Produced by: link to "Consultant"
Produced at: link to "Project"
Body
Abstract: file
Main text: list of files
Consultant
Project
Document 10
Title: C++ Training Module 1
Author: Hector Rizzoli
Tech area: <PL>
Purpose: training
Size in pages: 58
Date: 04-05-1991 • • •
Produced by: Consultant 12
Produced at: N/A
Abstract: file123.doc
Main text: <file124.doc>
Consultant 12
Name: Hector Rizzoli
Office: Greenbelt
Tech area: PL, Design, Testing
• • •

Visual Query Interface



Interface Prototype Evaluation

- Prototype included VQI, a simple data entry interface, and a small sample repository
- Objective was to evaluate basic interface design choices and attributes
- Study design included “think aloud” sessions with users and interviews
- Five sessions were conducted in May and June of 1999, each 1.5-2 hours

Evaluation Results

- Basic choices for search and retrieval interface and data model were well received
- Interface included some unexpected benefits
- Data entry interface was unacceptable
- Sample repository was too limited and inconsistent
- Evaluation was relatively low-cost and high-benefit

Conclusions

- The first results from a much-awaited line of research on developing infrastructure for Experience Factories
- Interface prototype of the Q-Labs EMS
- Prototype evaluation
- Future versions and extensions of EMS

Session 6: Panel Discussion

Lee Holcomb, NASA/CIO

Jerry Page, Computer Sciences Corporation

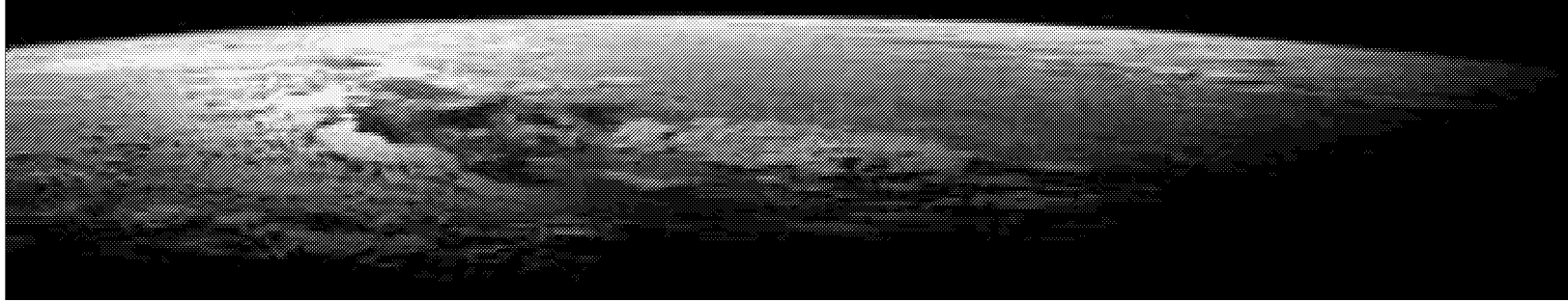
Mike Evangelist, National Science Foundation

Software Past, Present, and Future: View from the NASA CIO

NASA Software Engineering Workshop

December 2, 1999

Lee Holcomb



Software Past

- High-level language evolution (Fortran, Ada, C/C++, Java) … higher productivity, lower confidence
- Development and use of CMM
- Limited success of software reuse (NetLib)
- No silver bullet
- Hardware capacity (Moore's Law) outstrips software productivity
- Internet software development process (90-day time box)

Software Present

- Software development costs exceed plans and deliveries continue to be late
  - Costs often exceed plan by 50%, sometime by 100%
  - Most missions have a major software problem
  - Software intense projects are often 2 years late
- Software processes are still chaotic
- Software managers are not well trained
- Still no silver bullet
- Turnover of IT professionals is high

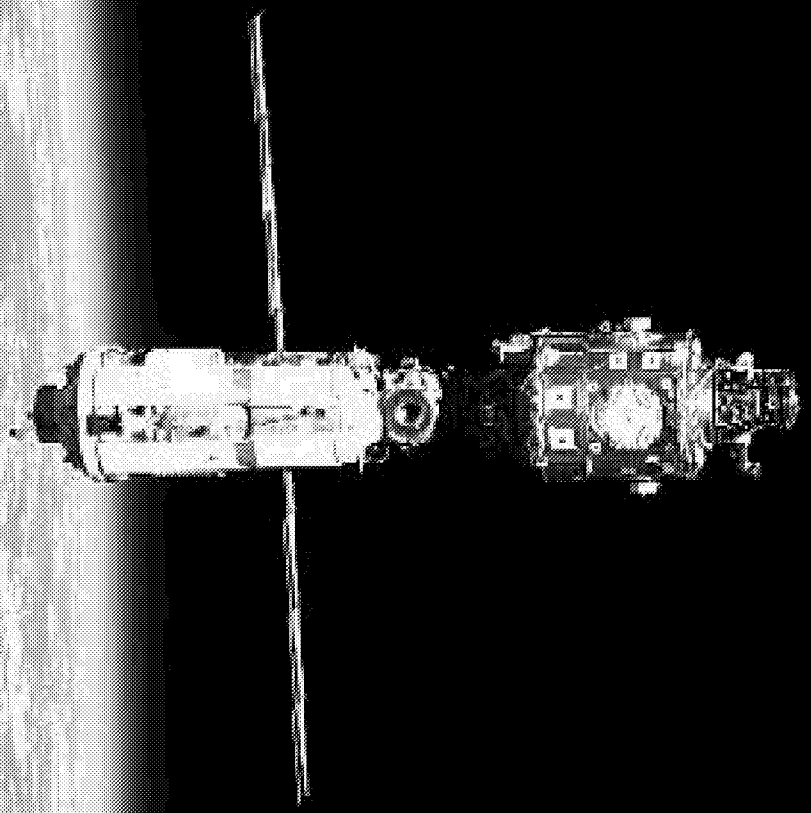
NASA's Largest Software Challenges

- Earth Observing System Data and Information System
  - NASA design, contractor developed, > Million Lines of Code (MLOC), COTS components
- Checkout Launch Control System
  - NASA design and development, > MLOC, COTS components
- Integrated Financial Management System
  - Contractor provided COTS >MLOC product

8330 Software Projects in Industry
Standish Group's 1994 Report

- 16 % were successful
  - In budget
  - On time
  - Met requirements
  - For large projects, only 9% were successful
- 53 % were "challenged"
  - Average 189% over budget
  - 222% late
  - 39% capabilities missing
- 31 % canceled during development

Software Future



- COTS
  - Market cycle yields poorly-tested, high-risk software
  - Complex software projects planned as all COTS evolve into COTS plus custom developed software
  - Customers with high-confidence applications will demand quality COTS
- Reuse/Formal Methods
  - Software reuse and formal methods have strong potential to improve quality and reduce cost
  - Reuse is still limited to well defined narrow functions
  - Formal methods have been limited to computer hardware or simple software applications
- Open source movement
  - Offers potential for thoroughly examined modular code
- Software development becomes a science

CMM Model : SEI Levels

1) Initial: Software process ad hoc, chaotic. Success depends on heroics.

2) Repeatable: Processes established to track cost, schedule, functionality

3) Defined: Process for management and engineering activities documented, standardized, and integrated

4) Managed:Detailed measures of software process and product quality collected

5) Optimizing: Continuous improvement

System Engineering Quality
Also Part of the Problem

- Most projects are now software intense
  - All modern system developments involve software
  - 90% of functionally is provided by software
- System engineering is the work above the software engineering layer
  - Requirements, architecture, risk management, integration, system testing, validation
- Quality system engineering is a prerequisite to quality software engineering
  - Must be partitioned into manageable elements
  - System engineers often have little software expertise

University Environment Trends
Will Increase the Problem in Software Engineering

- Undergraduate
  - Demand for graduates in computer science continues to exceed the supply of graduates
  - High starting salaries are increasing rate of dropouts
- Advanced computer science degrees
  - At one leading university computer science applicants dropped from 300 per year to 20 per year
  - Faculty members are being drawn into industry reducing the ability to train students
- Academic computer science research is declining

NASA Software Engineering Goals

1. Implement software engineering processes that are certified to Level 3 on the CMM scale for all NASA centers
   - Achieve level 3 in three years at 3 centers
2. Conduct software research to enable the development of large trusted software systems
4. Develop with universities a core curriculum for training software managers, software engineers, practitioners, and assurance personnel
5. Define and implement meaningful metrics

24th Annual Software Engineering Workshop

Software Past, Present, and Future: Views From Industry

Jerry Page

Industry Viewpoint: S/W Past

Significant Achievement: Process and Process as a Criterion

Then (1960s, 1970s)

- Cost and schedule unpredictable
- Error rates too high
- Deliveries made possible by heroes
- Heroes get burned out

Now (1990s, 2000s)

- Processes well defined, more standardized
- Processes are major discriminator in assessing contractor qualifications
- Processes adopted as common tool by development companies

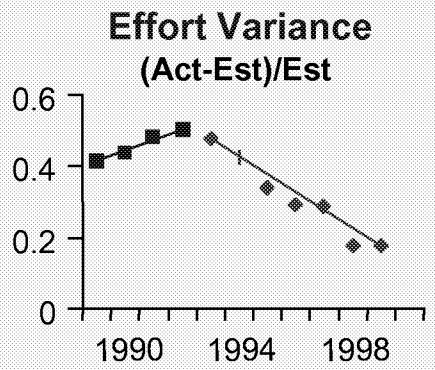
Significant Disappointment: Integrated Development Tools and Processes

Technologies have advanced in the small

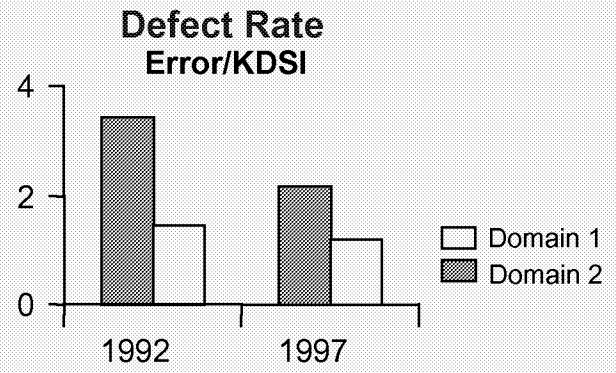
- When we attempt to solve large problems, new approaches easily hit limits
- Today's problems are no more complex than problems 20 years ago

Product-Driven Improvement Supported by Process*

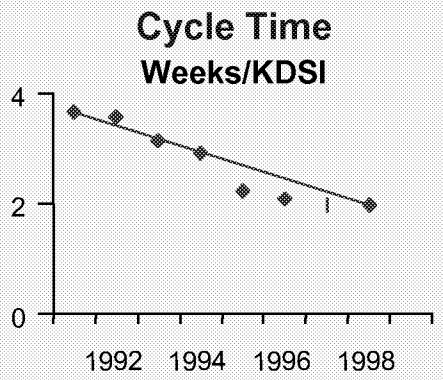
Estimate Better



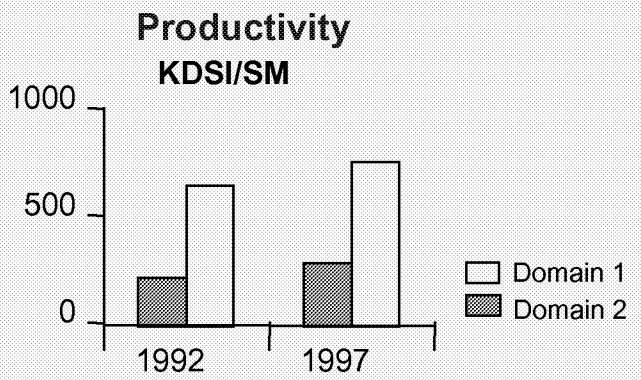
Improve Quality



Finish Faster



Reduce Costs



* SEAS Center data for over 90 projects 1988 - 1998

Industry Perspective: S/W Present

- **Good**

  A significant paradigm shift has facilitated rapid development of broader, newer applications

  - Driven by COTS, Reuse, 4GLs, Internet, ...

- **Bad**

  Shift has caused impact on recruiting, training, marketing, and has antiquated established processes

  - Process paradigm has been outpaced by the development paradigm

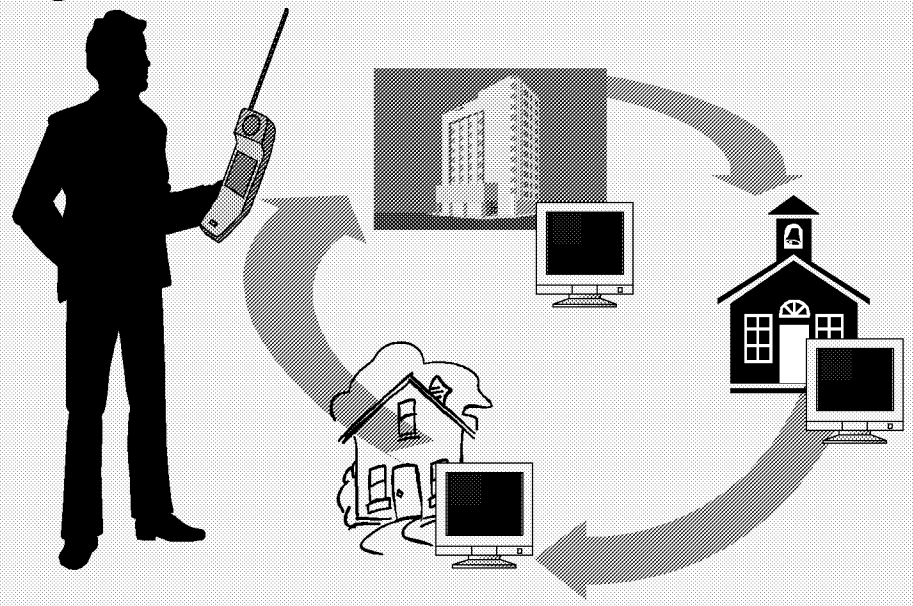
- **Challenge**

  Research and development to define new processes and tools

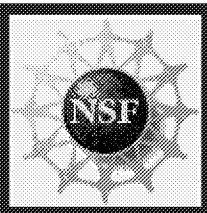
Industry Perspective: S/W Future

Significant Breakthroughs

- **Enabling Technologies**
  - Voice-activated programming
  - Wireless communication
- **Challenges**
  - Voice pattern recognition
  - Security concerns
  - Retraining of staff



Computer and Information Science and Engineering

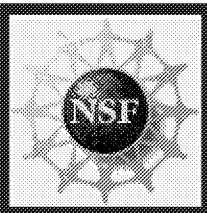


"ERROR ACC127. Couldn't find file WZN3.MDA. What do you want to do?":

The Future of Software?

Michael Evangelist
Director
Computer-Communications Research Division
National Science Foundation

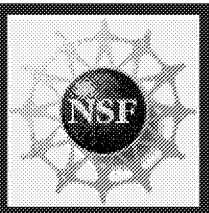
Past Hits and Misses



Greatest success: Our $400 billion industry

Greatest disappointment: Help that research (industry) has given to industry (research)

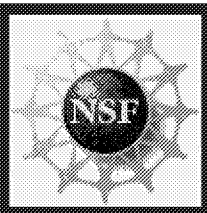
Software Research Today



Good: Increasing emphasis on empirical investigation and validation

Bad: Increasing resistance to empirical investigation and validation

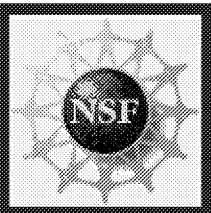
It Might Not Get Easier



5-10 years: "Commodization" of computing — from millions to billions of users

- will the billions read error messages?
- moving applications to servers changes "customer" concept

It Might Not Get Easier, II



15-20 years: Radically new computing "platforms" built with nanotechnologies

- how to create software for 100 million interconnected processors?
- where does the error message appear when the software is in your knee?

Session 7: Inspections

Edward Weller, Bull HN Information Systems

Al Florence, MITRE

Amarjit Singh Marjara, Cap Gemini AS

24th Annual Software Engineering Workshop
Dec 1-2, Goddard Space Flight Center

Quantitative Methods ***Do*** Work

Edward F. Weller
Fellow, Software Process
Bull HN Information Systems
13430 N. Black Canyon
Phoenix, AZ 85029

Tele: (602) 862-4563
Fax: (602) 862-4288
e-Mail: e.weller@bull.com

Quantitative Methods *Do* Work

Quantitative methods, including statistical process control, can be effective tools for predicting and evaluating product quality during development and test. The data analysis and conclusions from applications of quantitative methods, including statistical process control, to two projects that were major components of a software release to Bull HN Information System's GCOS 8 Operating System, will show how these techniques were effective and useful. During development and test, we used the release quality predictions as one of the project metrics. We found that analysis of inspection and test results using SPC techniques helped us predict (perhaps understand is a better word) the release quality and the development processes controlling the release quality. We were able to answer the question "Can we ship this product?" with data rather than guesswork.

Inspections have been used in GCOS 8 development since 1990[1]. The process is stable and provides data used by project management[2]. Our goal in the current release was to use defect density during development and test as input to predicting the post ship product quality with reasonable assurance. We are aware of the problems with using defects to predict failures (Adams[3], Fenton and Pfleeger[4]), but in the absence of other data or usage based testing results, this was what we had to evaluate release quality.

Prediction: Stable versus Unstable Processes

Predicting the future behavior of a process cannot be done unless the process itself is stable. This is a reason for using statistical methods. A variety of techniques can be used to evaluate the underlying process stability. Control charts can be used to calculate upper and lower control limits (UCL and LCL). Processes that stay within limits and do not exhibit other indications of lack of control can be assumed to be "controlled processes". This implies several things about the process:

- Past performance can be used to predict future performance within the control limits
- Process capability relative to a customer specification can be determined

Estimating Defect Injection

Previous inspection process and product data were evaluated and estimates were made for:

- Defect injection rates
- Defect removal rates (inspection effectiveness[I])
- Defects entering unit test

Prior inspection, test, and post ship defect history was used to estimate the defect injection rate. This is a potential area for applying SPC. With enough data, you can establish ranges for defect injection rates, accuracy of size estimates, and inspection

[I] Inspection effectiveness is the percentage of major defects removed in each inspection phase, or total defects removed in inspections, divided by the total number of defects in the product at the time of the inspection. Since the total number of defects discovered is never known until a product is retired from use, effectiveness is always an estimate, but one that changes very slightly after a product is shipped, assuming reasonable post ship quality levels.

removal effectiveness. For these projects we did not have sufficient data samples to do this, so we based our estimates on specific product and project history.

The size, defect injection rates, and prior inspection data were used to develop a defect injection and removal profile.

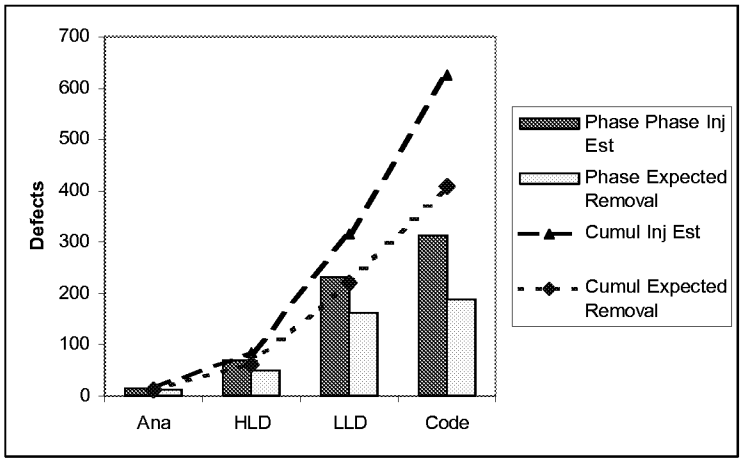


Figure 1 – Initial Defect Injection and Removal Estimate

Inspection Data Analysis

On these two projects the first opportunity to apply SPC was during code inspections. On one project, the work was divided into two parts; the creation of a product feature, and the revision of existing code. A histogram of preparation rates in lines of code per hour is shown in Figure 2.

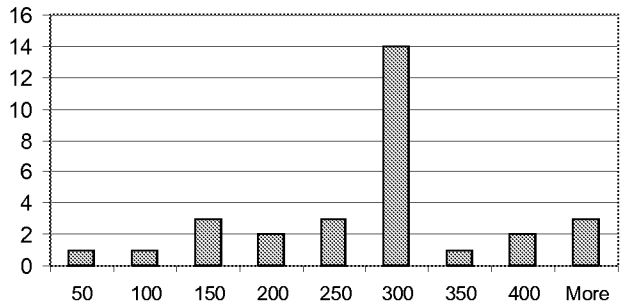


Figure 2 – Preparation Rate Histogram

Outliers were examined and eliminated when special causes of variation were discovered. I also compared the preparation rate distribution to the inspection rate, shown in Figure 3.

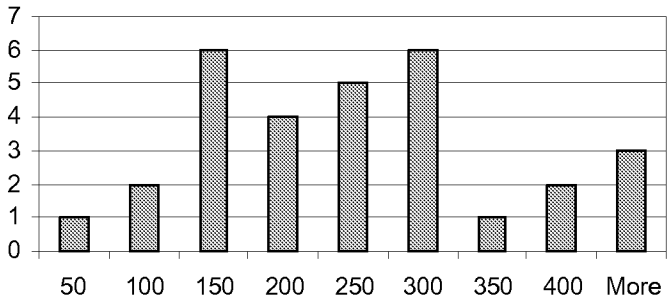


Figure 3 – Inspection Rates for 30 inspections

This bimodal distribution was caused by two types of code, "new", and changed. Figure 4 shows these 2 classes separately.

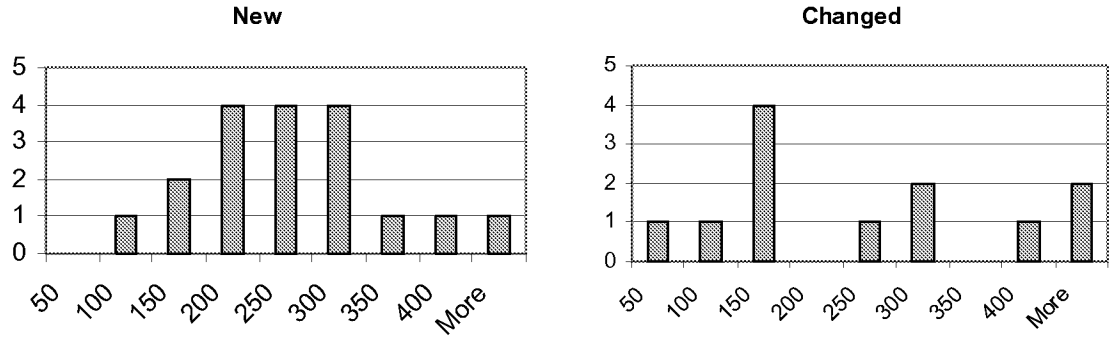


Figure 4 – New vs Changed Inspection Rates

I expect new code inspections to be "better behaved' than changes to existing code. (Many inspections of modified code are small in size, causing preparation and inspection rates to have a larger variance. Knowledge of the changed (old) code inspected may also have a wider variance than the new code). The separate views in Figure 4 are typical of much of the inspection data I investigate. The new code approximates a normal distribution as closely as you may see with real data.

A control chart for this data with special causes removed showed a well controlled process:

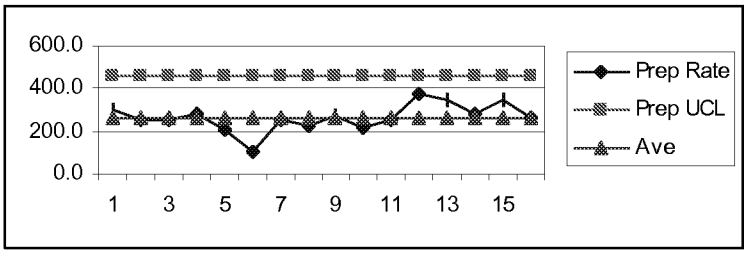


Figure 5 – Preparation Rate with Outliers Removed

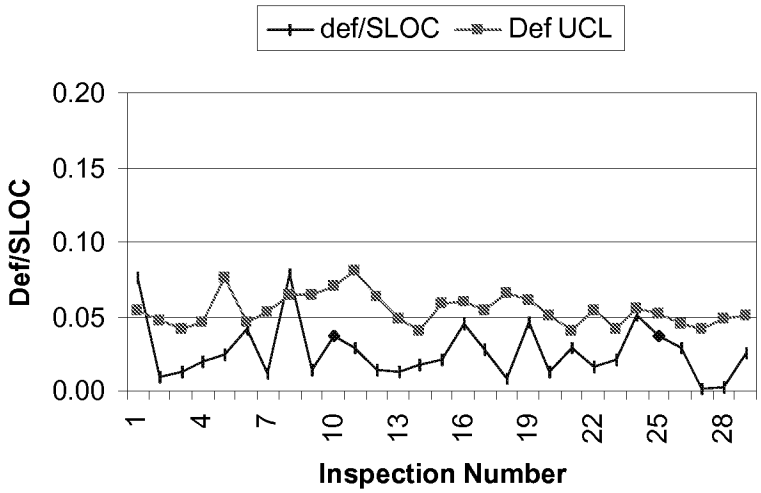


Figure 6 – Defect Density Control Chart[II]

What have we learned about this product and its contribution to the system release? With two exceptions, the inspection process seems to be well controlled. The outliers were investigated (as were other inspection meetings) to understand how well the inspection process was performed. In this case, the outliers were for inspections of changed code, so these outliers were evaluated as caused an assignable cause, and the defect data was within control limits.

Once we were reasonably confident the inspection process was controlled, we developed defect depletion curves for the projects and the system release.

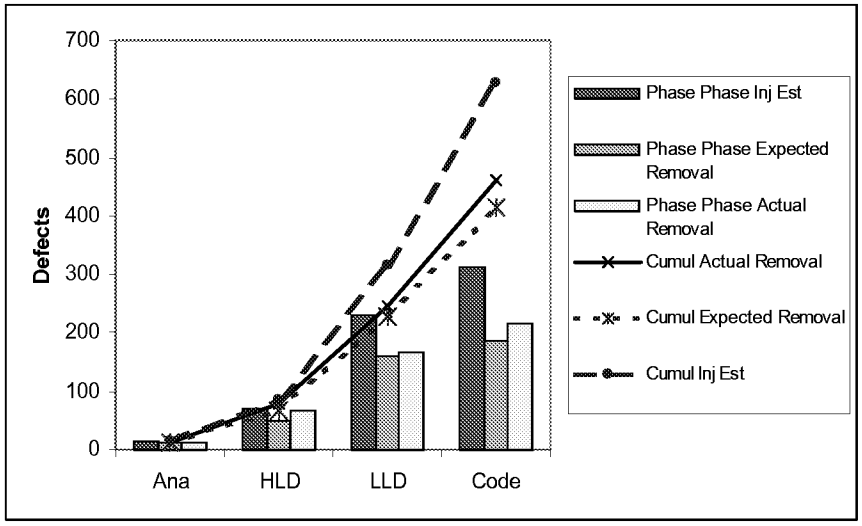


Figure 7 – Defect Depletion at End of Code Inspection

[II] The lower control limits cannot be less than zero, although for convenience the LCL was plotted on this chart as calculated. Once you verify the data is above the LCL, for possible values of LCL, it is probably better to delete this line from the chart.

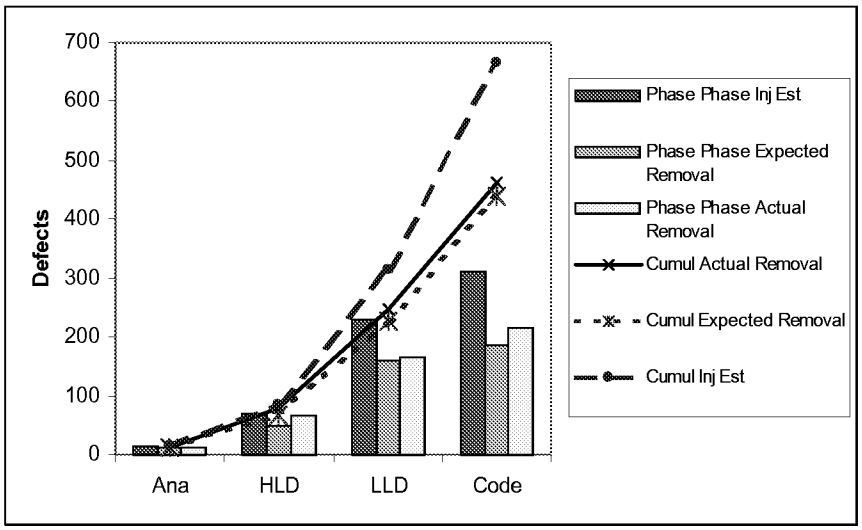


Figure 8 – Replotted Defect Depletion with New Size Estimate

Unit and Integration Test

Both projects kept accurate records of defects found during Unit and Integration test. Both projects developed test objective matrices and developed test plans and specifications, so we had some expectation to remove defects more effectively than the 30-50% "norm" often quoted in the industry.

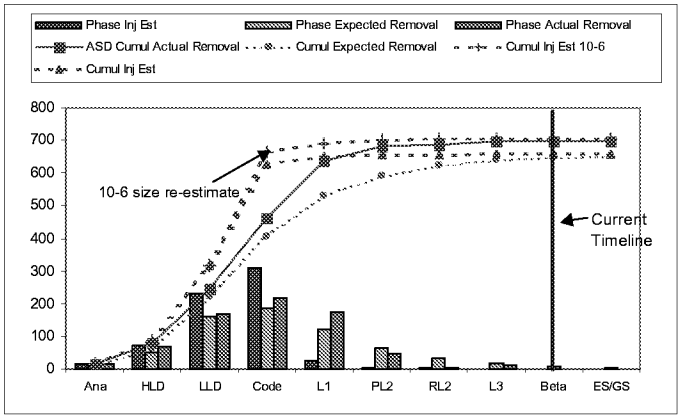


Figure 9 – Project One Defect Depletion

Figure 9 shows project one, as it was about to enter System Test (this chart is used in our monthly project review as well as the weekly team meetings). It shows the re-estimate for the number of defects injected. Note the defect removal in Unit Test was higher than estimated and that subsequently in the two phases of Integration Test a small number of defects were removed. Without accurate defect removal data from Unit Test these low numbers would be of more concern with respect to product quality. The Current Timeline is indicated to show the furthest stage where the project defect removal is happening.

This analysis continued through System Test as shown in Figure 10.

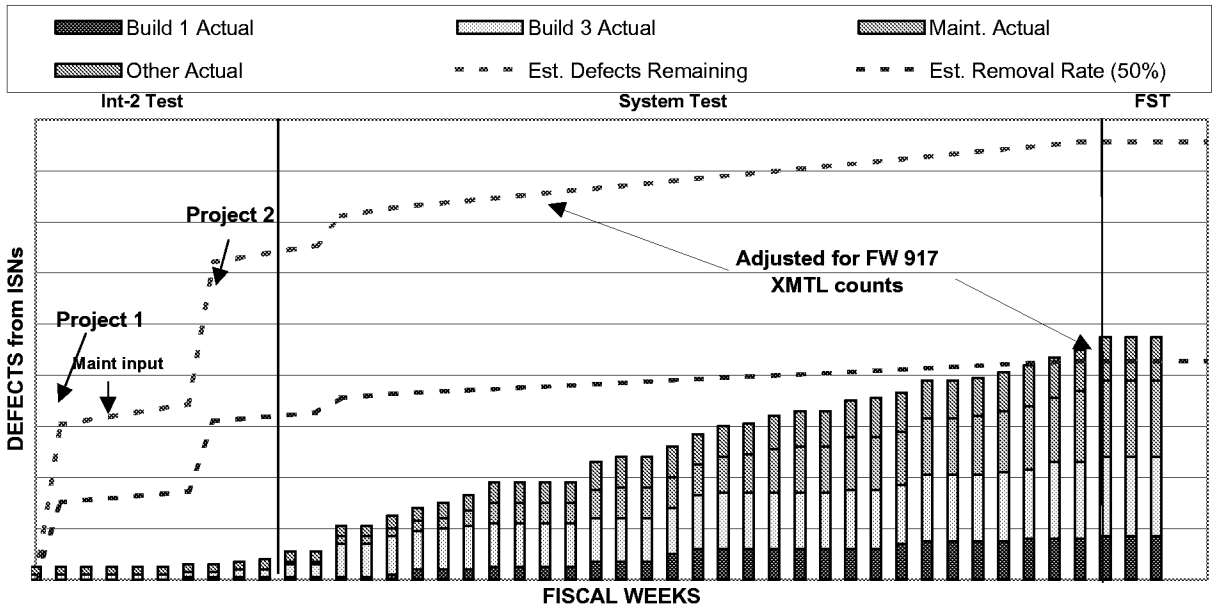


Figure 10 – System Test

Conclusions

You should ask two questions about any metric or analysis technique:

- Is it *useful*? Does it provide information that helps make decisions?
- Is it *useable*? Can we reasonably collect the data and do the analysis?

We found that the knowledge we gained about product quality and the processes used to develop these products gave a definite "Yes" to both these questions.

---

[1] E.Weller, "Lessons Learned from 3 Years of Inspection Data", *IEEE Software*, Sept 1993
[2] E.Weller, " Using Metrics to Manage Software Projects", *IEEE Computer*, Sept 1994
[3] E. Adams, "Optimizing Preventive Service of Software Products", *IBM Journal of Research & Development*, Jan 1984
[4] N. Fenton and S. Pfleeger, Software Metrics, PWS Publishing Company, 1997, pp 344-348

Quantitative Methods *Do* Work

Ed Weller

Fellow, Software Process

Bull Worldwide Information Systems

Phoenix, AZ

e.weller@bull.com

Worldwide Information Systems

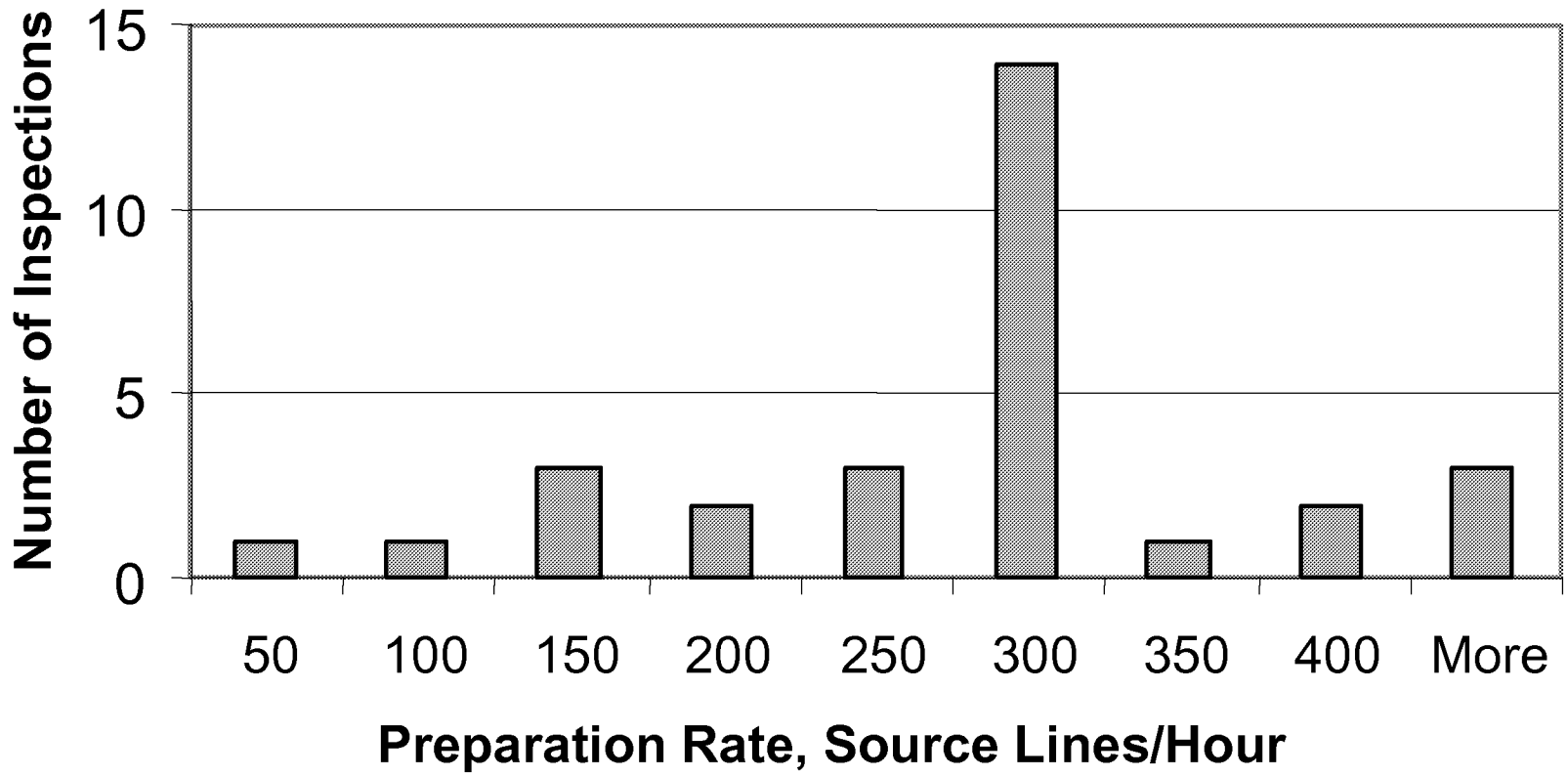
Bull

Why Did We Decide to Use Quantitative Methods?

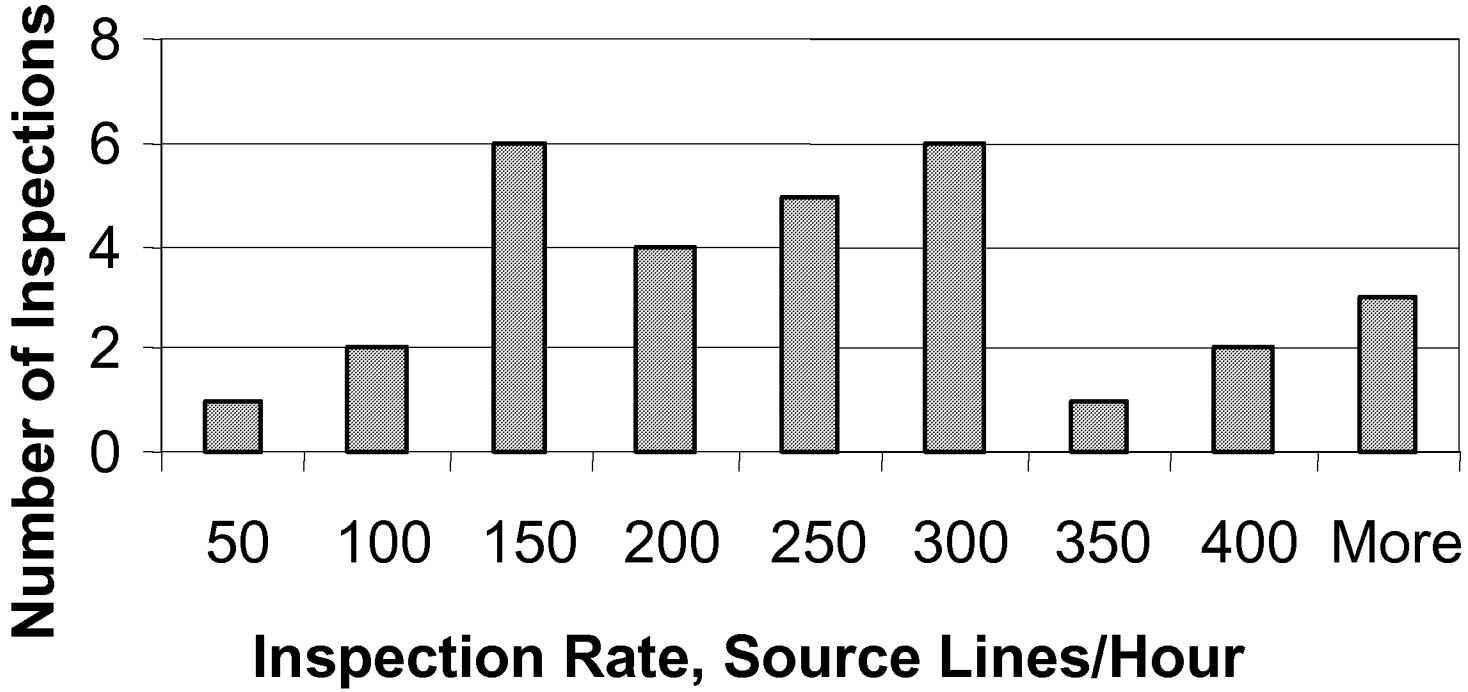
- **Neat stuff?**
- **Intellectually challenging?**
- **To make Business Decisions**
  - **Will we meet quality goals?**
  - **How well is test progressing?**
  - **How well are we inspecting?**
  - ***Can we ship this product?***

If using quantitative methods doesn't meet a business need, don't use it!

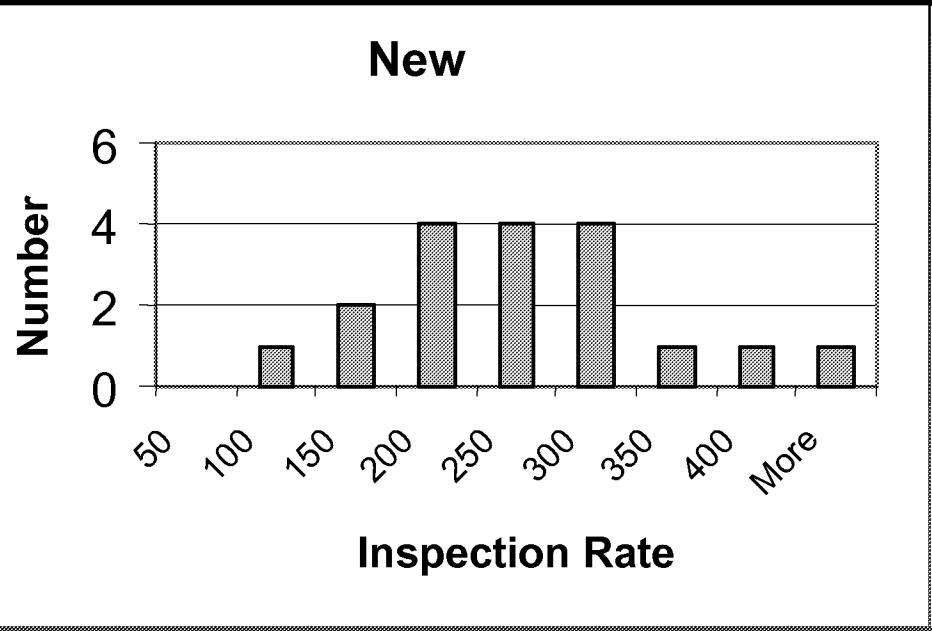
How Well Are We Doing Inspections? - 1

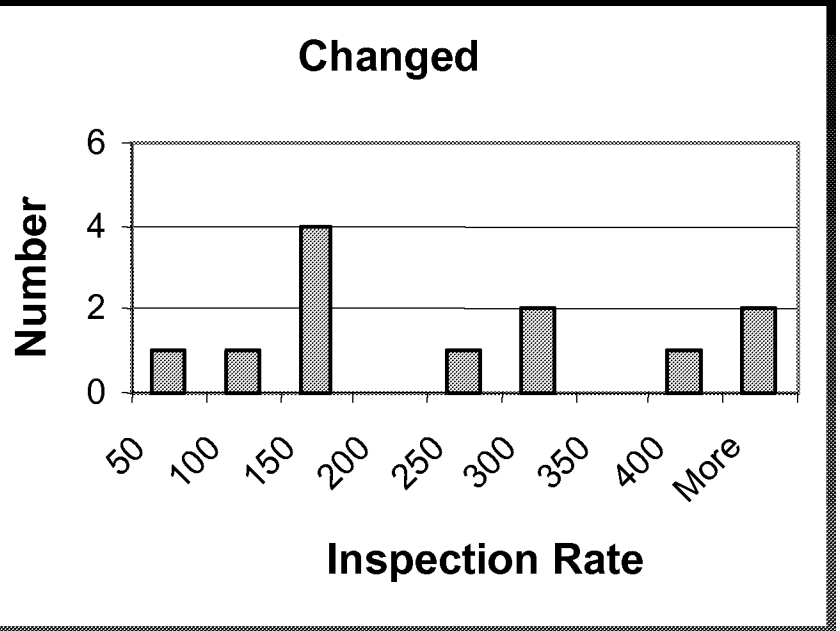


How Well Are We Doing Inspections? - 2



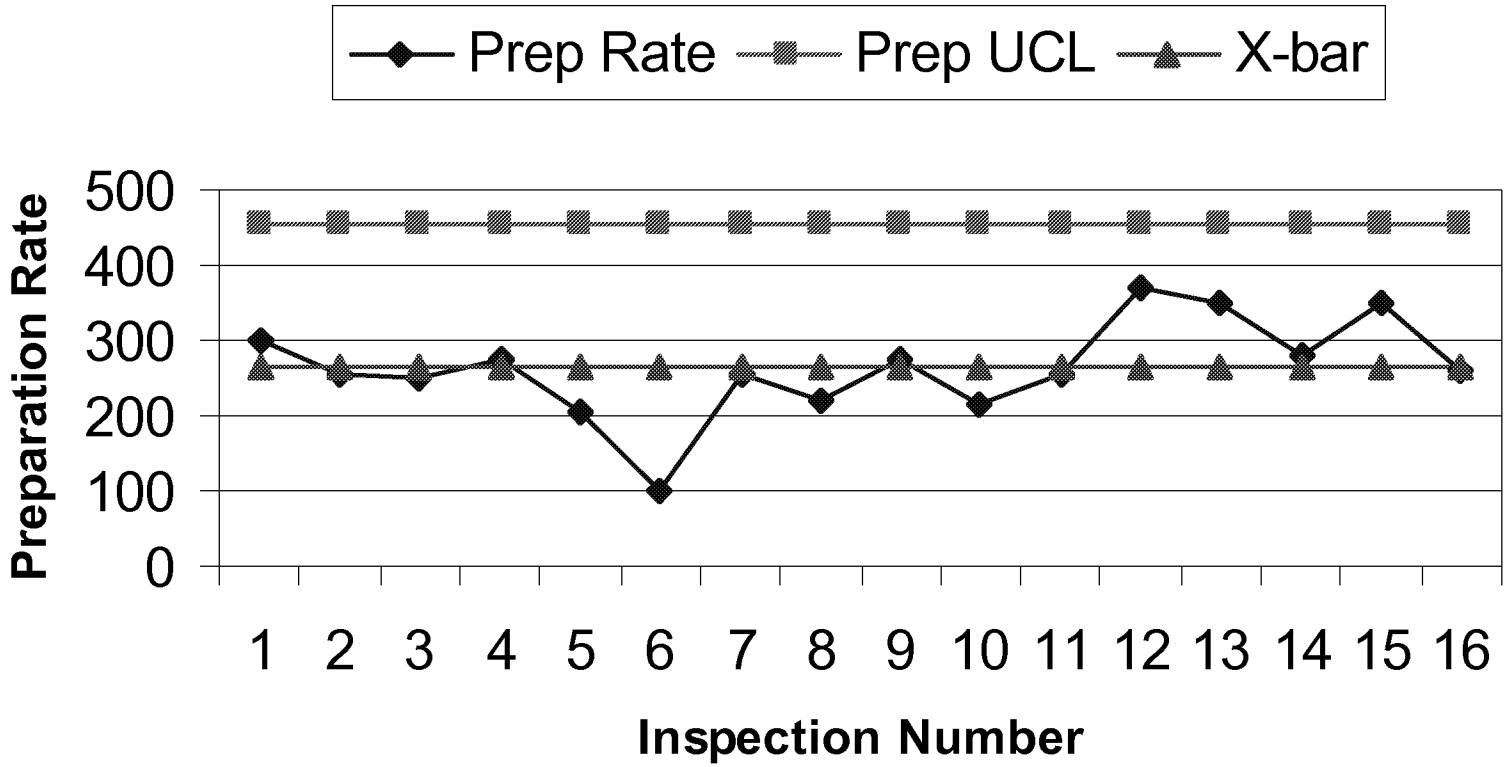
How Well Are We Doing Inspections? - 3





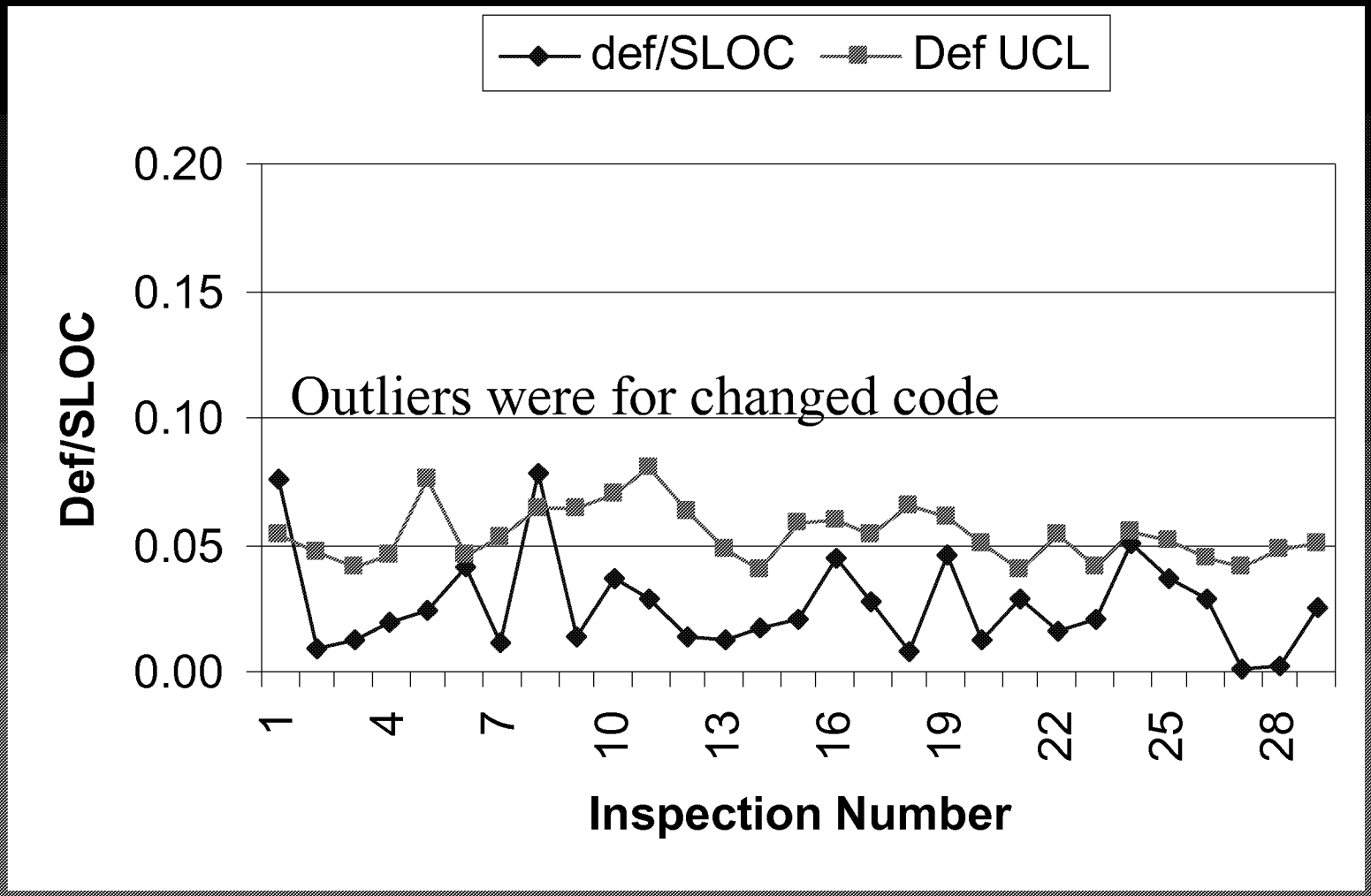
Comparing "New" modules with "Changed" modules

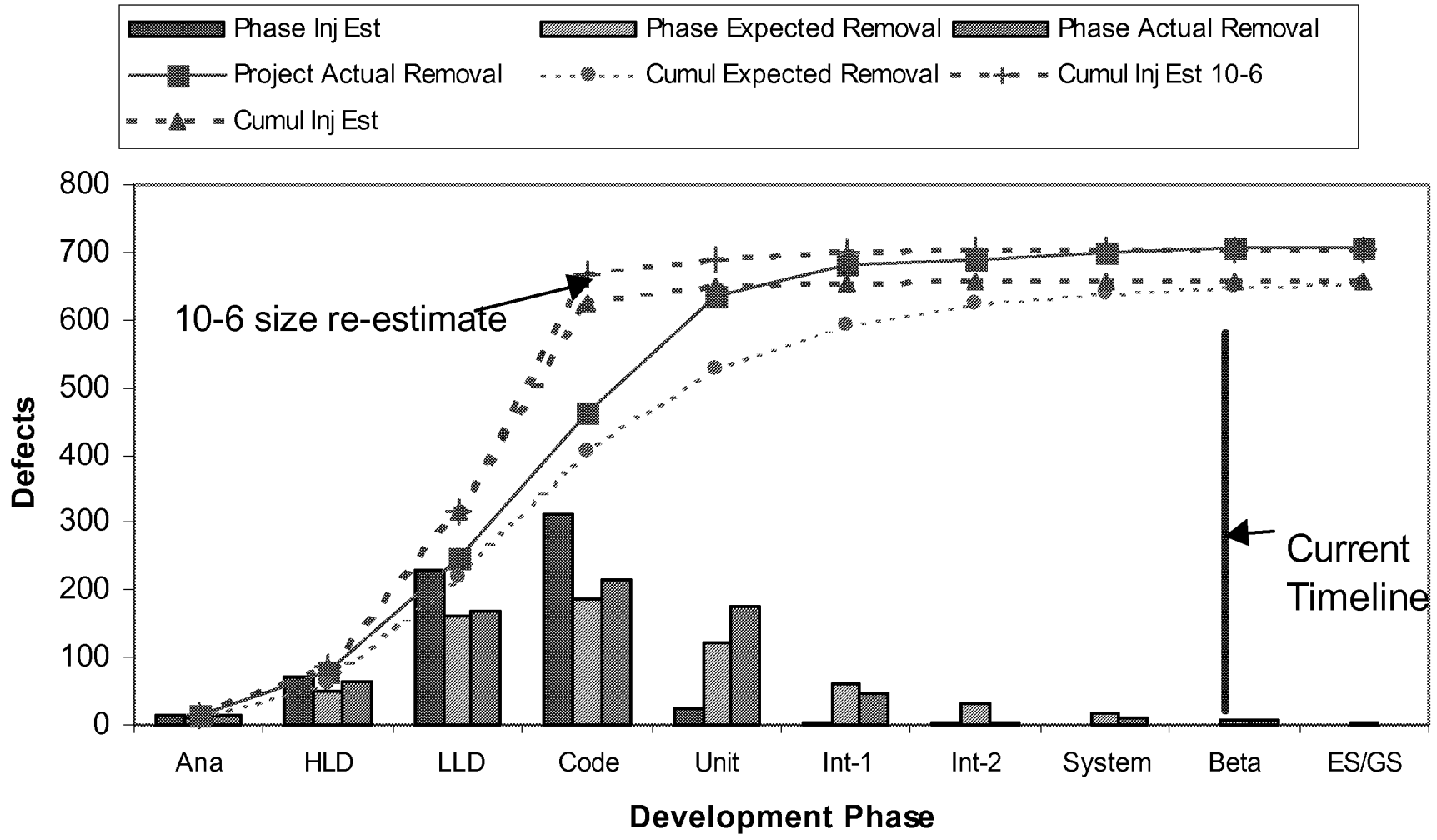
How Well Are We Doing Inspections? - 4



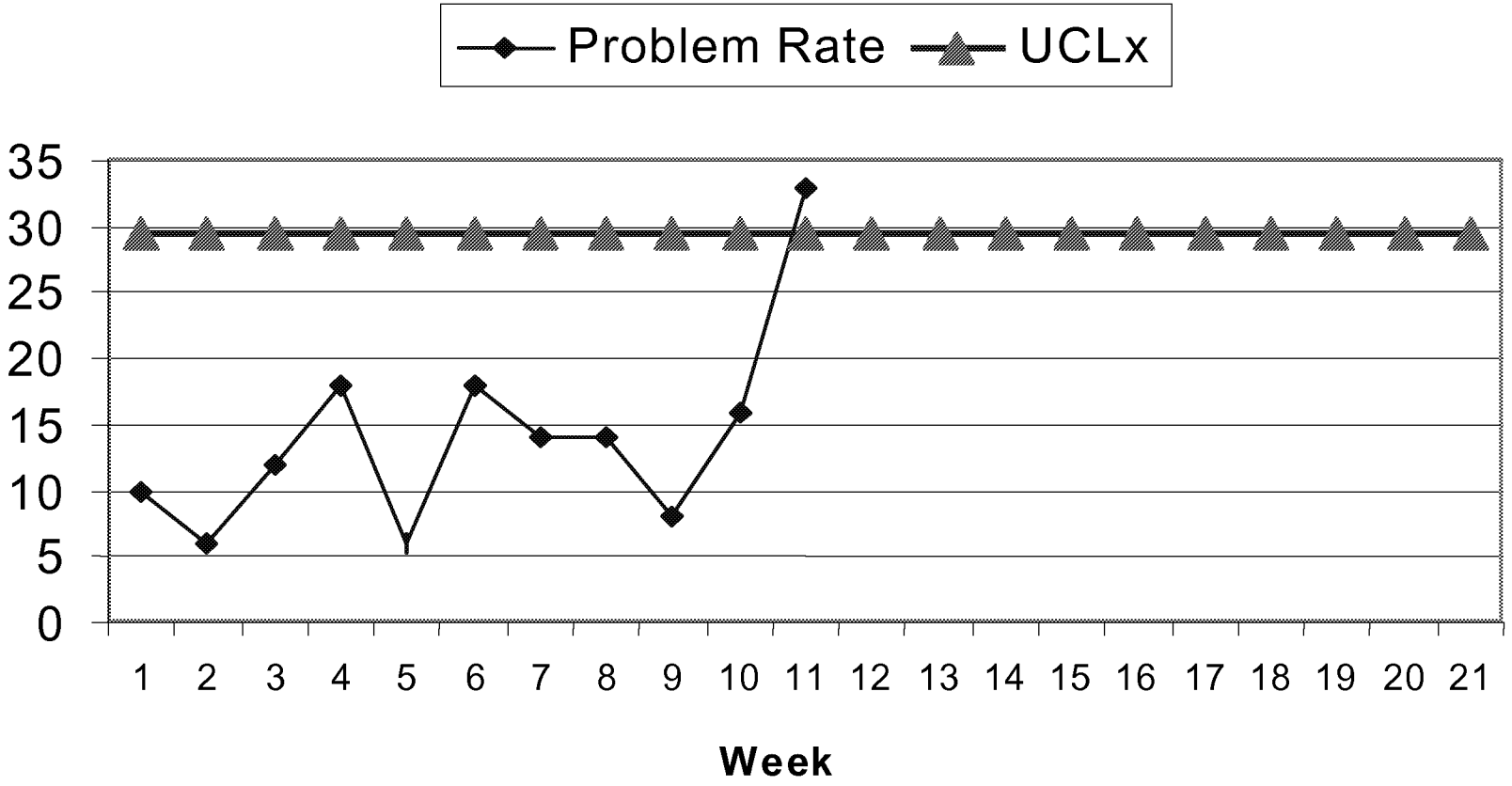
Two special causes of variation removed
(caused by work product variation)

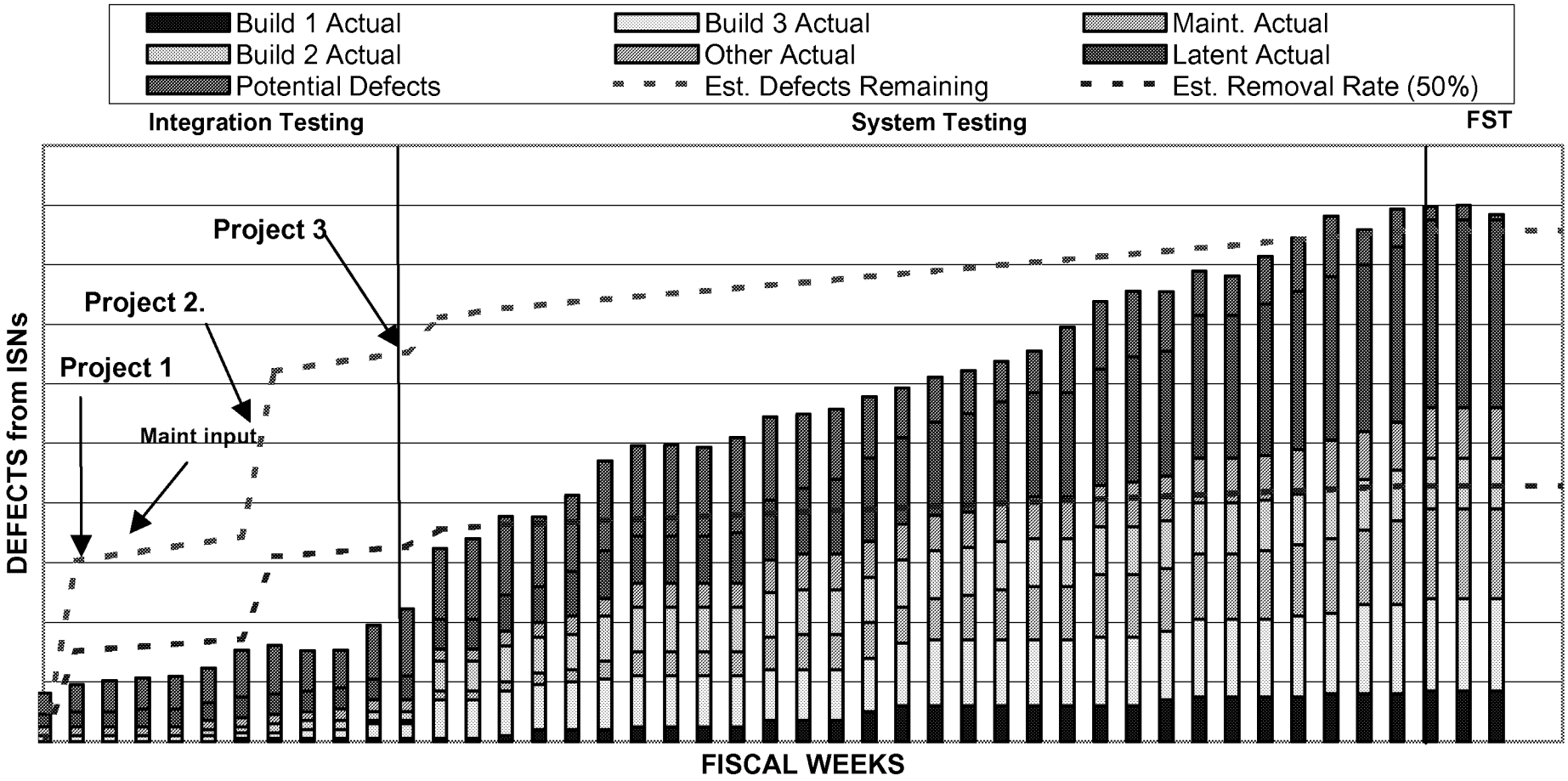
How Well Are We Doing Inspections? - 5





How Well Are We Doing in Test?





How do you know if this detection rate is good or bad?

Evaluating the Product

- **Was the inspection process a controlled process?**
- **How good was the product when it entered test?**
- **How well was it tested?**
- **What was the residual defect rate?**
- ***Can we ship?***

SEI CMM Level 4 Quantitative Analysis
Real Project Examples

Al Florence
December, 1999
MITRE Corporation

The views expressed are those of the author and do not reflect the official policy or position of the MITRE Corporation

Key Words

- Quantitative Process Management
- Software Quality Management
- Defect Prevention
- Quantitative Analysis
- Statistical Process Control
- Control Charts

Abstract

The Software Engineering Institute's (SEI) Software (SW) Capability Maturity Model (CMM) Level 4 Quantitative Analysis leads into SW-CMM Level 5 activities. Level 4 Software Quality Management (SQM) Key Process Area (KPA) analysis, which focuses on product quality, feeds the activities required to comply with Defect Prevention (DP) at Level 5.[1] Quantitative Process Management (QPM) at Level 4 focuses on the process which leads to Technology Change Management (TCM) and Process Change Management (PCM) at Level 5. At Level 3, metrics are collected, analyzed and used to status development and to make corrections to development efforts, as necessary. At Level 4, metrics are quantitatively analyzed to control process performance of the project and to develop a quantitative understanding of the quality of products to achieve specific quality goals. At Level 5, the Level 4 analysis is used, as appropriate, to investigate and incorporate new processes and technologies and for the prevention of defects.

This paper presents the application of Statistical Process Control (SPC) in accomplishing the intent of SQM and QPM and applying the results to DP. Real project results are used to demonstrate the use of SPC as applied in a software setting. Presented are the processes that the author formulated, launched and conducted on a large software development effort. The organization had obtained SW-CMM Level 3 compliance and was pursuing Level 4 and Level 5. All Level 4 and Level 5 processes were installed and conducted on the project over a period of time. The main quantitative tool used was Statistical Process Control utilizing control charts. The project analyzed life cycle metrics collected during development for requirements, design, coding, integration, and during testing. Defects were collected during these life cycle phases and were quantitatively analyzed using statistical methods. The intent was to use this analysis to support the project in developing and delivering high quality products and at the same time using the information to make improvements, as required, to the development process.

Introduction

This introduction presents an overview of SPC and why it is applied to software. It presents a review of the Level 4 KPAs and Defect Prevention at Level 5. Next, Level 4 quality goals and plans to meet those goals are described followed by some real project examples in applying SPC to real project data.

Control Charts

Figure 1 shows a control chart and demonstrates how control charts are used for this analysis.[3] According to the normal distribution, 99% of all normal random values lie within +/-3 standard deviations from the norm, 3-sigma.[3] If a process is mature and under statistical process control, all events should lie within the upper and lower control limits. If an event falls out of the control limits the process is said to out of statistical process control and the reason for this anomaly needs to be investigated for cause and the process brought back under control.

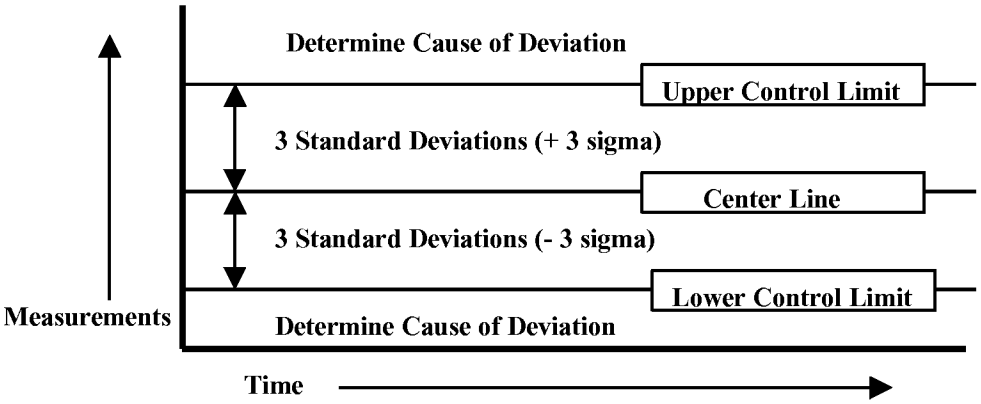


Figure 1. Control Chart

Control charts are used because they separate signal from noise, so when anomalies occur they can be recognized. They identify undesirable trends and point out fixable problems and potential process improvements. Control charts show the capability of the process, so achievable goals can be set. They provide evidence of process stability, which justifies predicting process performance.

Control charts use two types of data: variables data and attributes data. Variables data are usually measurements of continuous phenomena. Examples of variables data in software settings are elapsed time, effort expanded, and memory/CPU utilization. Attributes data are usually measurements of discrete phenomena such as number of defects, number of source statements, and number of people. Most measurements in software used for SPC are attributes data. It is important to use the correct data on a particular type of control chart.[3]

Quantitative Analysis Flow

Figure 2 shows the Level 4 Quantitative Analysis process flow for Software Quality Management and for Quantitative Process Management.[1]

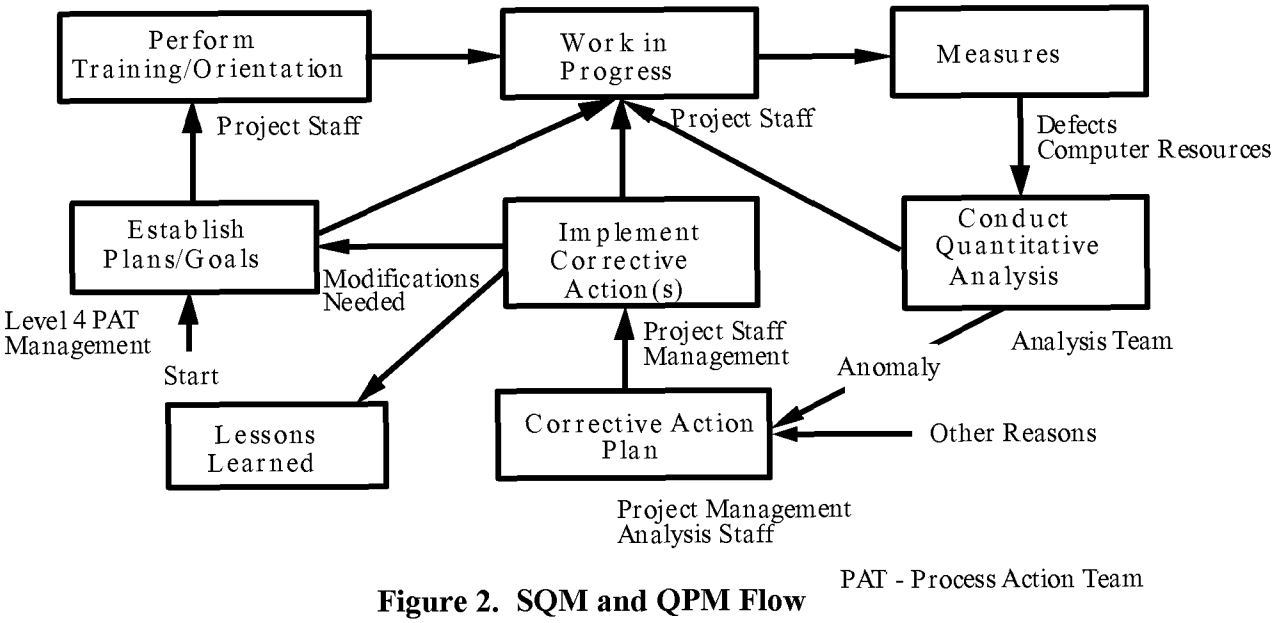


Figure 2. SQM and QPM Flow

When conducting quantitative analysis on project data the results can be used for both Software Quality Management and for Quantitative Process Management. If the data analyzed are defects detected, the intent is to reduce the defects during the activities that detected the defects throughout development, thus satisfying SQM. When out of statistical control conditions occur, the reason for the anomaly is investigated and the process brought back under control which satisfies QPM.

Defect Prevention Flow

Figure 3 shows the Level 5 Defect Prevention process flow.[1]

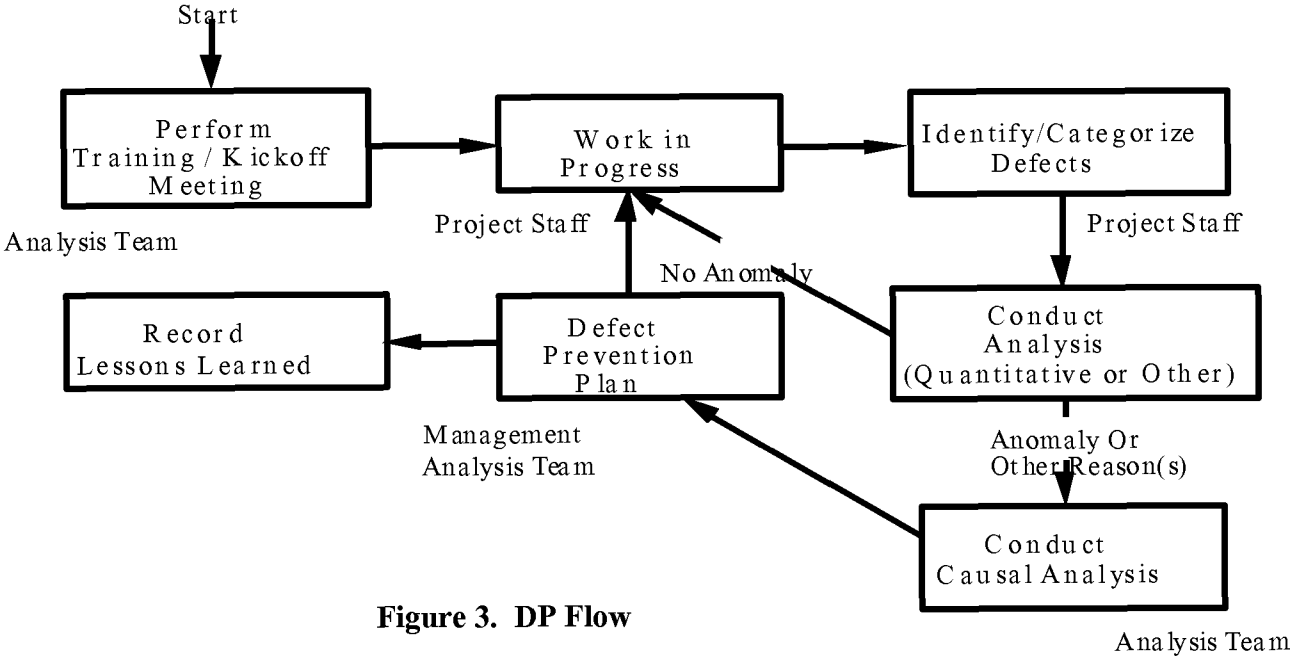


Figure 3. DP Flow

Defects can occur during any life cycle activity against any and all entities. How often do we see requirements that are without problems or schedules that are adequate or management that is sound? Defect Prevention activities are conducted on any defects that warrant prevention. Defect prevention techniques can be applied to a variety of items:

- Project Plans
- Project Schedules
- Standards
- Processes
- Procedures
- Project Resources
- Requirements
- Documentation
- Quality Goals
- Design
- Code
- Interfaces
- Test Plans
- Test Procedures
- Technologies
- Training
- Management
- Engineering

Level 4 Feeds Level 5

Figure 4 shows how data collection, analysis and management from Level 4 activities lead to the activities at Level 5 of Defect Prevention, Technology Change Management, and Process Change Management KPAs.[5]

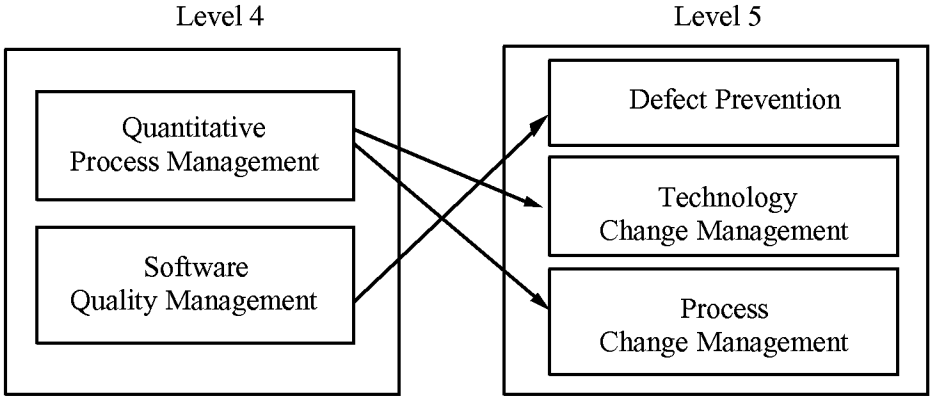


Figure 4. Level 4 and Level 5 Paths of Influence

Quantitative Process Management, which focuses on the process, leads to making process and technology improvements while Software Quality Management, which focuses on quality, leads to preventing defects.

Level 4 Goals and Plans

The CMM requires that Level 4 goals, and plans to meet those goals, be based on the processes implemented, that is, on the processes' proven ability to perform.[1] Goals and plans must also reflect contract requirements. As the project's process capabilities and/or contract requirements change, the goals and plans may need to be adjusted.

The project that this paper is based on had the following key requirements:

- Timing - subject search response in less than 2.8 seconds 98% of time
- Availability - 99.86% 7 days, 24 hours (7/24)

These are driving requirements that constrain hardware and software architecture and design. To satisfy these requirements, the system needs to be highly reliable and with sufficiently fast hardware.

Goals

The planned quality goals are:

- Deliver a near defect free system
- Meet all critical computer performance goals

Plans

The plans to meet these goals are:

- Defect detection and removal during
  - Requirements peer reviews
  - Design peer reviews
  - Code peer reviews
  - Unit tests
  - Thread tests
  - Integration and test
  - Formal tests
- Monitoring of critical computer resources
  - General purpose million instructions per second (MIPS)
  - Disc storage read inputs/outputs per second (IOPS) per volume
  - Write IOPS per volume
  - Operational availability
  - Peak response time
  - Server loading

Quantitative Analysis Examples

The following are real examples from the project discussed above applying SPC to real data over a period of two years.

Example 1

Table 1 shows raw data collected at code peer reviews over a period of months. Each sample represents a series of peer reviews over several weeks. The "units" are units of code and the "SLOC" is the number of source lines of code (SLOC) review for that sample. The "defects" are the number of defects detected for that sample normalized to 1000 lines of code in the last column.

Table 1. Code Peer Review Defects

Sample	Units	SLOC	Defects	Defects/KSLOC
1. Mar 1998	6	515	15	29.12
2. Apr 1998	10	614	16	26.06
3. Apr 1998	7	573	7	12.22
4. Apr 1998	7	305	7	22.95
5. Apr 1998	4	350	21	60
6. Apr 1998	3	205	2	9.76
7. Apr 1998	8	701	11	15.69
8. May 1998	3	319	3	9.40
Totals	**76**	**3582**	**72**	

The formulas for constructing the control chart follow.[3] The control chart used is a U-chart.

- Defects/KSLOC = Number of Defects * 1000/SLOC reviewed per sample (calculated for each sample). These are plotted as Plot.
- CL = Total Number of Defects/Total number of KSLOC reviewed * 1000
- a(1) = SLOC reviewed/1000 (calculated for each sample)
- UCL = CL+3(SQRT(CL/a(1)) (calculated for each sample)
- LCL = CL-3(SQRT(CL/a(1)) (calculated for each sample)

The defects per 1000 lines of code is the plot on the chart. The center line (CL) is an average while a(1) is a variable calculated for each sample. The upper control limit (UCL) and the lower control limit (LCL) are also calculated for each sample. The calculations are shown in Table 2. Whenever the LCL is negative, it is set to zero.

Table 2. Calculations for Code Peer Review Defects

Sample	Plot	CL	UCL	LCL	a(1)
1. Mar 1998	29.13	20.1	38.84	1.36	0.515
2. Apr 1998	26.06	20.1	37.27	2.96	0.614
3. Apr 1998	12.22	20.1	37.87	2.333	0.573
4. Apr 1998	22.96	20.0	44.45	0	0.305
5. Apr 1998	60	20.1	42.84	0	0.35
6. Apr 1998	9.76	20.1	49.80	0	0.205
7. Apr 1998	15.71	20.1	36.16	4.04	0.701
8. May 1998	9.40	20.1	43.91	0	0.319

The control chart is shown in Figure 5.

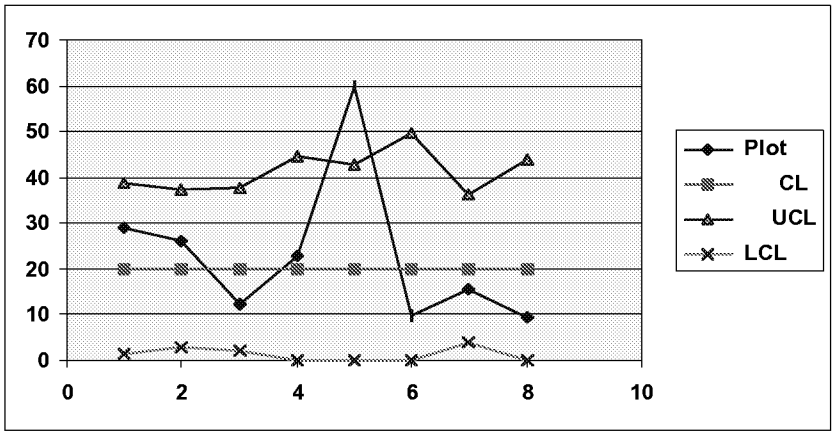


Figure 5. Control Chart for Code Peer Review Defects

An anomaly occurred in the fifth sample. Causal analysis revealed that data for that sample were for database code, all others were applications code. Control charts require similar data for similar processes, i.e., apples to apples analogy. The database sample was removed and the data charted again as shown in Figure 6.

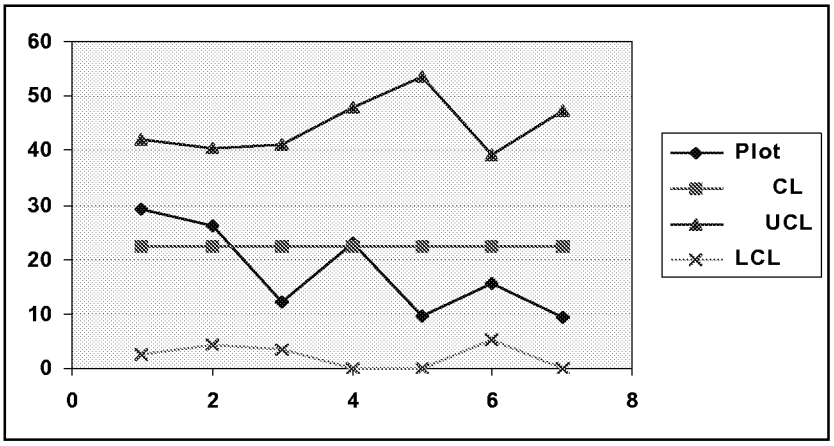


Figure 6. Control Chart without Database Defects

The process in now under statistical process control. The root cause is that data gathered from dissimilar activities cannot be used on the same statistical process on control charts. Data from design cannot be combined with data from coding. The process for database design and code is different from that used for applications design and code as are the teams and methodologies. The defect prevention is against the process of collecting data for SPC control charts.

Example 2

Table 3 shows raw data collected during code peer reviews.

Table 3. Code Peer Review Defects

Sample	Units	SLOC	Defects	Defects/KSLOC
1. Feb 1997	17	1705	62	36.36
2. Mar 1997	18	1798	66	36.70
3. Mar 1997	15	1476	96	65.04
4. Mar 1997	19	1925	57	29.61
5. Mar 1997	17	1687	78	46.24
6. Apr 1997	18	1843	66	35.81
Totals	**104**	**10434**	**425**	

The calculations are shown in Table 4.

Table 4. Calculations for Code Peer Review Defects

Sample	Plot	CL	UCL	LCL	A(1)
1. Feb 1997	36.4	40.73	55.4	26.09	1.7
2. Mar 1997	36.7	40.73	55.01	26.45	1.8
3. Mar 1997	65	40.73	56.49	24.97	1.5
4. Mar 1997	29.6	40.73	54.53	26.93	1.9
5. Mar 1997	45.2	40.73	55.47	25.99	1.7
6. Apr 1997	35.8	40.73	54.84	26.63	1.8

The control chart is shown in Figure 7.

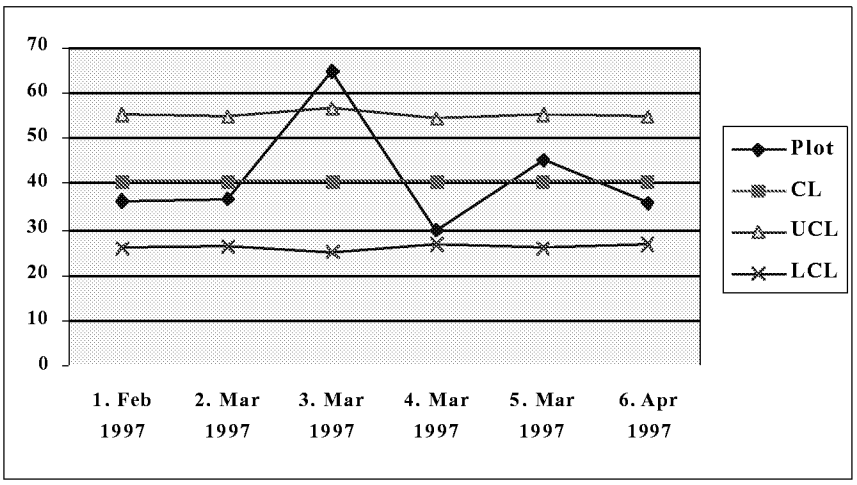


Figure 7. Control Chart for Code Peer Review Defects

The process is out of statistical process control in the third event. Causal analysis revealed that this was caused when the project introduced coding standards and many coding violations were

injected. The root cause is lack of knowledge of the coding standards and the defect prevention is to provide training whenever a new process or technology is introduced.

Example 3

During integration thread tests, the defects were categorized against the test plan, test data, code logic, interfaces, standards, design, and requirements. Defects against these attributes are shown in Table 5.

Table 5. Thread Test's Defects

Samples	Test Plan	Test Data	Logic	Interface	Standards	Design	Requirements
1	2	6					
2		10					
3	1	9	3				
4	2	1	13				
5		1	7				
6		10	14				
7		4	2				
8		28					
9						2	
10			6				
11	1	1					
12		10					
13		9	1				
14		6	1	1			
15		5	7				
16		2	1				
Totals	**6**	**102**	**55**	**1**		**2**	

Bar chars were used in Figure 8 to show defects discovered during integration thread tests.

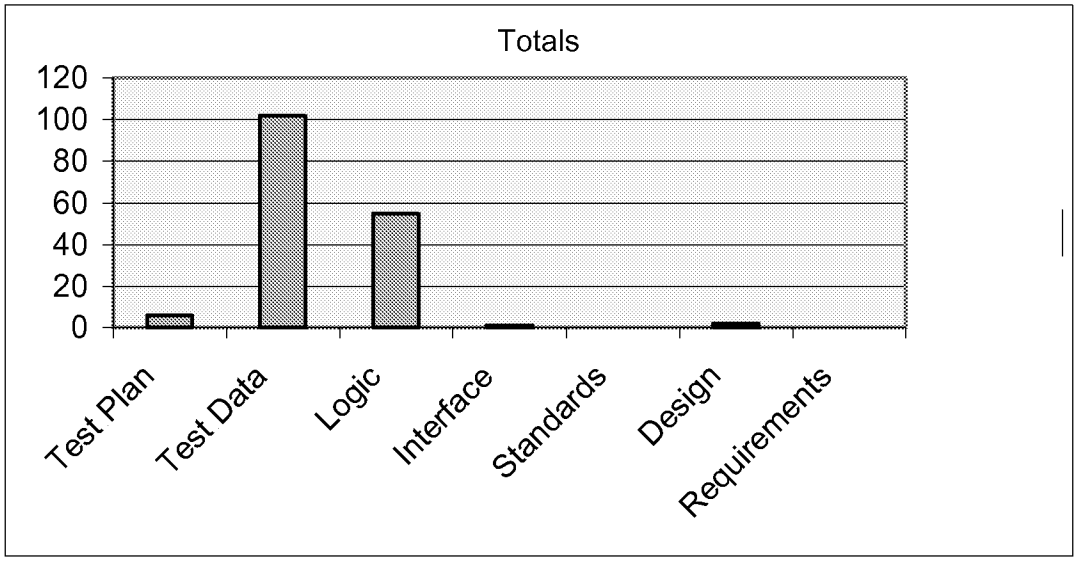


Figure 8. Thread Test Defects Bar Chart

Test data would not be expected to have the majority of defect. The root cause was that the test data in the test procedures had not been peer reviewed. The defect prevention is to peer review the test procedures and the test data.

Example 4

During preliminary design and prior to acquiring hardware, a simulated performance model was used to monitor critical computer resources. Figure 9 shows some results of monitoring general purpose MIPS.

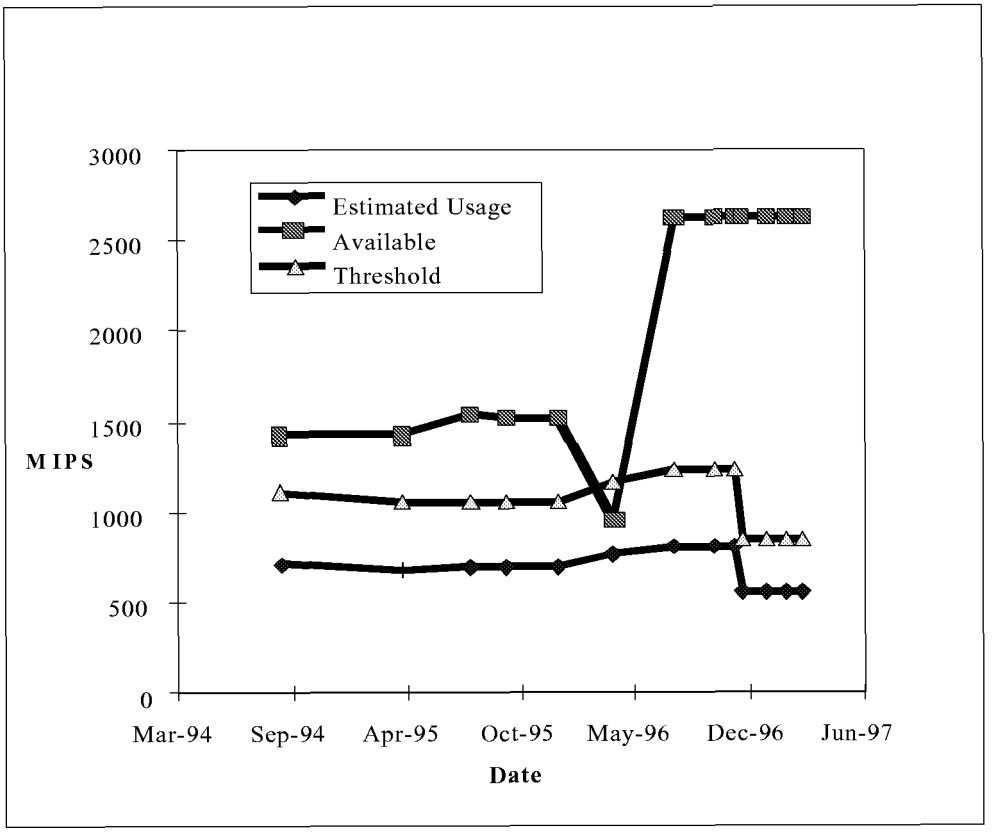


Figure 9. General Purpose MIPS

Around November 1995 many new requirements were added to the system and the architecture's MIPS threshold was threatened because of increased computations. In May 1996 additional MIPS were added to the hardware design and the problem was corrected.

Conclusion

Statistical process control and the use of control charts can be effectively used in a software setting. SPC can identify undesirable trends and point out fixable problems and potential process improvements. Control charts can show the capability of the process, so achievable goals can be set. They can provide evidence of process stability, which can justify predicting process performance.

References
and
Suggested Reading

1. Mark C. Paulk, Bill Curtis, Mary Beth Chrissis, Charles V. Weber, February 1993, *Capability Maturity Model for Software*, V1.1, Software Engineering Institute
2. John H. Baumert, Mark S. McWhinney, 1992, *Software Measures and the Capability Maturity Model*, Software Engineering Institute
3. William A. Florac, Robert E. Park, Anita D. Carleton, SEI, April, 1997, *Practical Software Measurement: Measuring for Process Management and Improvemen*t, Software Engineering Institute
4. Thomas Pyzdek, 1984, *An SPC Primer*, Quality America, Inc.
5. Ron Radice , 1997, *Getting to Level 4 in the CMM*, The 1997 SEI Software Engineering Process Group Conference, San Jose, CA
6. Anita Carleton, Mark C. Paulk, 1997, *Statistical Process Control for Software*, The 1997 Software Engineering Symposium, Pittsburgh, PA
7. David S. Chambers & Donald J. Wheeler, 1995, *Understanding Statistical Process Control*, SPC Press
8. *Juran's Quality Control Handbook*, 1988, 4th Edition, McGraw-Hill Book Company
9. Donald J. Wheeler, 1993, *Understanding Variation, The Key to Managing Chaos*, SPC Press
10. Donald J. Wheeler, 1995, *Advanced Topics in Statistical Process Control*, SPC Press
11. W. Edwards Deming, November 1975, *On Probability As a Basis For Action*, The American Statistician, Vol. 29, No.4 (146-152)
12. Gerald J. Hahn & William Q. Meeker, February 1993, *Assumptions for Statistical Inference*, The American Statistician, Vol. 47, No.1 (1-11)
13. Watts S. Humphrey, September 1997, *Managing the Software Process*, SEI Series in Software Engineering, Addison-Wesley Publishing Company

MITRE

24th Annual Software Engineering Workshop December 1999

Capability Maturity Model Level 4 Quantitative Analysis

Al Florence

The views expressed are those of the author and do not reflect the official policy or position of the MITRE Corporation

Agenda

- Capability Maturity Model (CMM) Level 4/5 overview
- Level 4
  - Quantitative Process Management (QPM)
  - Software Quality Management (SQM)
  - Statistical process control (SPC)
  - Quantitative analysis
- Level 5
  - Defect Prevention (DP)

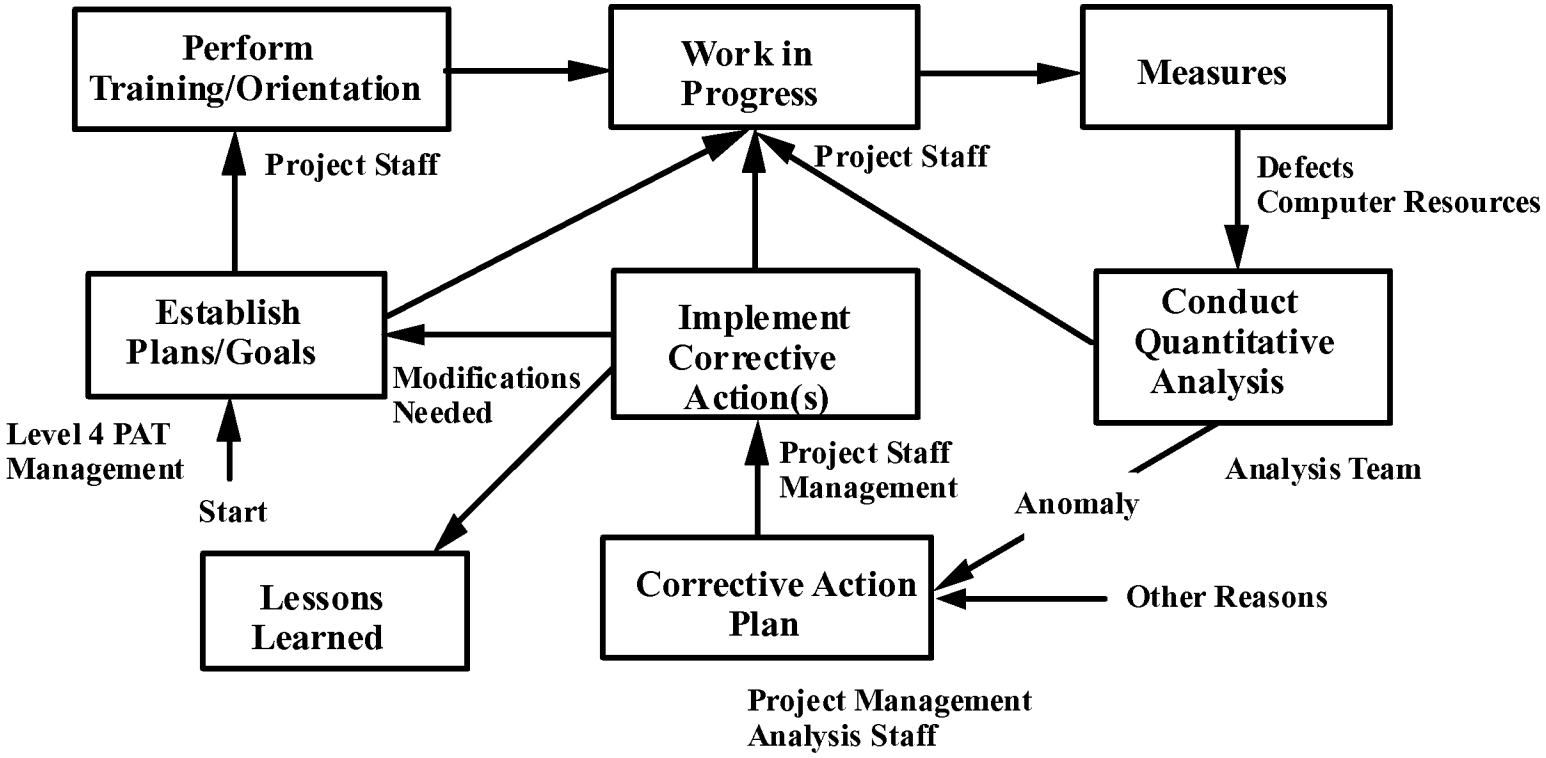
Level 4 - Managed

- **Quantitative Process Management - Process Focus**
  - To control the process performance of the software project quantitatively
- **Software Quality Management - Product Focus**
  - To develop a quantitative understanding of the quality of the project's software products and achieve specific quality goals

Level 5 Optimizing

- **Defect Prevention**
  - **To identify the cause of defects and prevent them from recurring**
- **Technology Change Management**
  - **To identify new technologies (i.e., tools, methods, and processes) and transition them into the organization in an orderly manner**
- **Process Change Management**
  - **To continually improve the software process used in the organization with the intent of improving software quality, increasing productivity, and decreasing the cycle time for product development**

Level 4 QPM/SQM Process



PAT - Process Action Team

Level 4 Plans/Goals

- Level 4 goals, and plans to meet those goals, are based on the project's proven capability to perform
- Goals and plans must also reflect contract requirements
- As the project's capabilities and contract requirements change, the goals and plans may need to be adjusted

Plans/Goals Example - Actual Project

- Project's key driving requirements
  - Timing - subject search response in less than 2.8 seconds 98% of time
  - Availability - 99.86% 7 days, 24 hours (7/24)
- These are driving requirements that constrain hardware & software architecture & design
- To satisfy these requirements, the system needs to be highly reliable and hardware robust
- The Planned Quality Goals are:
  - Deliver a near defect free system
  - Meet all Critical Computer Performance Goals

Plans/Goals Example

- Plans are to detection and removal defects during:
  - Requirements peer reviews
  - Design peer reviews
  - Code peer reviews
  - Unit tests
  - Thread tests
  - Integration and test
  - Formal test

Plans/Goals Example

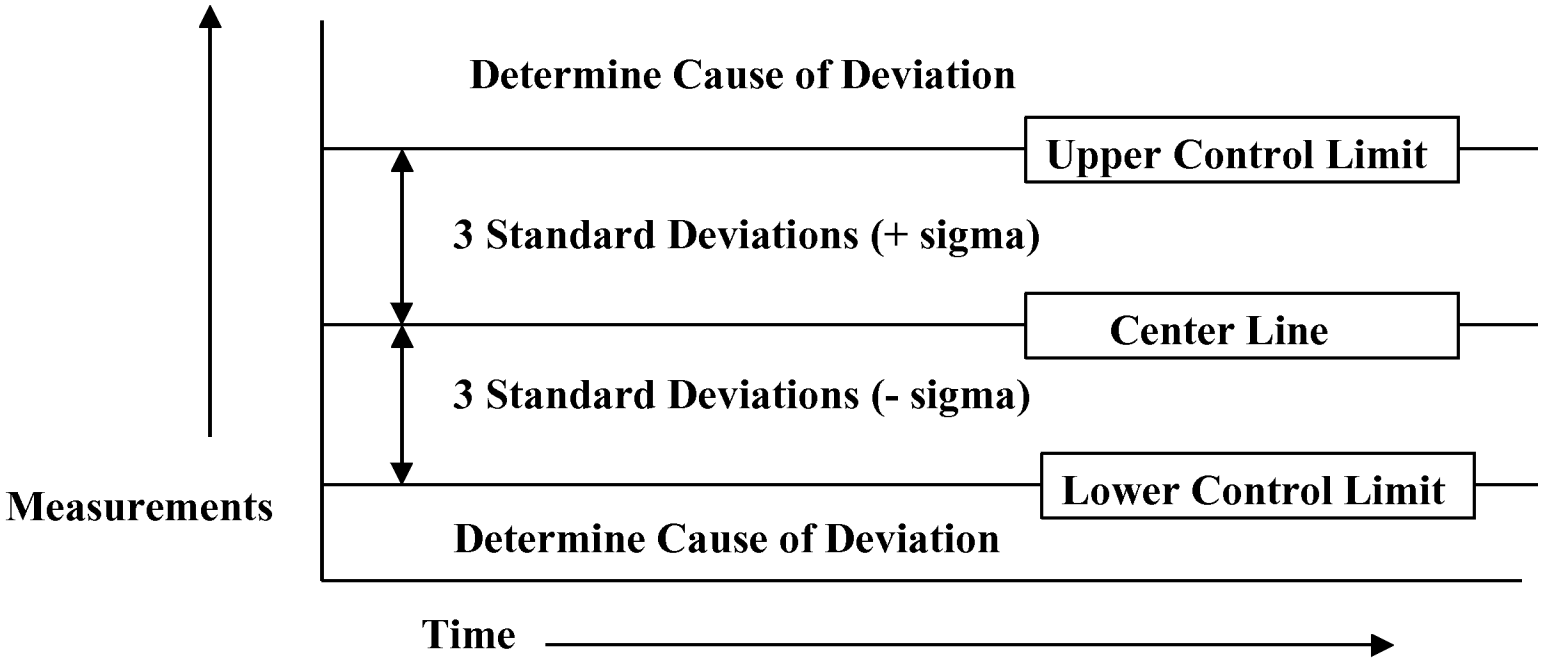
- Plans are to monitor Critical Computer Resources
  - General Purpose Million Instructions Per Second (MIPS)
  - Disc Storage Read Inputs/Outputs Per Second (IOPS) Per Volume
  - Write IOPS Per Volume
  - Operational Availability
  - Peak Response Time
  - Server Loading

Statistical Tools

The following tools were used to conduct the quantitative analysis

- **Statistical Process Control (SPC) - SPC using control charts & Bar Charts**
- **Performance Model - To monitor critical computer resource**

Statistical Process Control Charts



According to the Normal Distribution, 99% of all normal random values lie within +/-3 standard deviations from the norm, that is, 3 sigma

Statistical Process Control Charts

Why Control charts:

- **Separate signal from noise, so when anomalies occur they can be recognized**
- **Identify undesirable trends, they point out:**
  - **Fixable problems**
  - **Potential process improvements**
- **Show the capability of the process, so achievable goals can be set**
- **Provide evidence of process stability, which justifies predicting process performance**

Variables Data and Attributes Data

- **Variables Data**
  - Usually measurements of continuous phenomena
    - Length, weight, height, volume, voltage, torque
  - In software settings
    - Elapsed time, effort expanded, memory/cpu utilization
- **Attributes Data**
  - Usually measurements of discrete phenomena (counts)
    - Number of defects, number of source statements, number of people
  - Most measurements in software used for SPC are attributes data

Variables Data and Attributes Data

- **Control Limits**
  - Control limits for variables and attributes data are computed in quite different ways
- **Control Charts for Variables Data**
  - X bar
  - Range charts
  - XmR Charts
- **Control Charts for Attributes Data**
  - u charts
  - Z charts
  - XmR Charts

Other Quantitative Methods

- Check Sheets
- Run Chart
- Histogram
- Scatter Diagram
- Pareto Chart
- Flow Chart
- Fishbone Diagram

SPC Example - Code Peer Reviews

Raw data collected for code peer reviews

Sample	*Units*	*SLOC*	*Defects*	*Defects/KSLOC*
1. Feb 1997	17	1705	62	36.36
2. Mar 1997	18	1798	66	36.71
3. Mar 1997	15	1476	96	65.04
4. Mar 1997	19	1925	57	29.61
5. Mar 1997	17	1687	78	46.26
6. Apr, May	18	1843	66	35.81
Totals	104	10434	425	

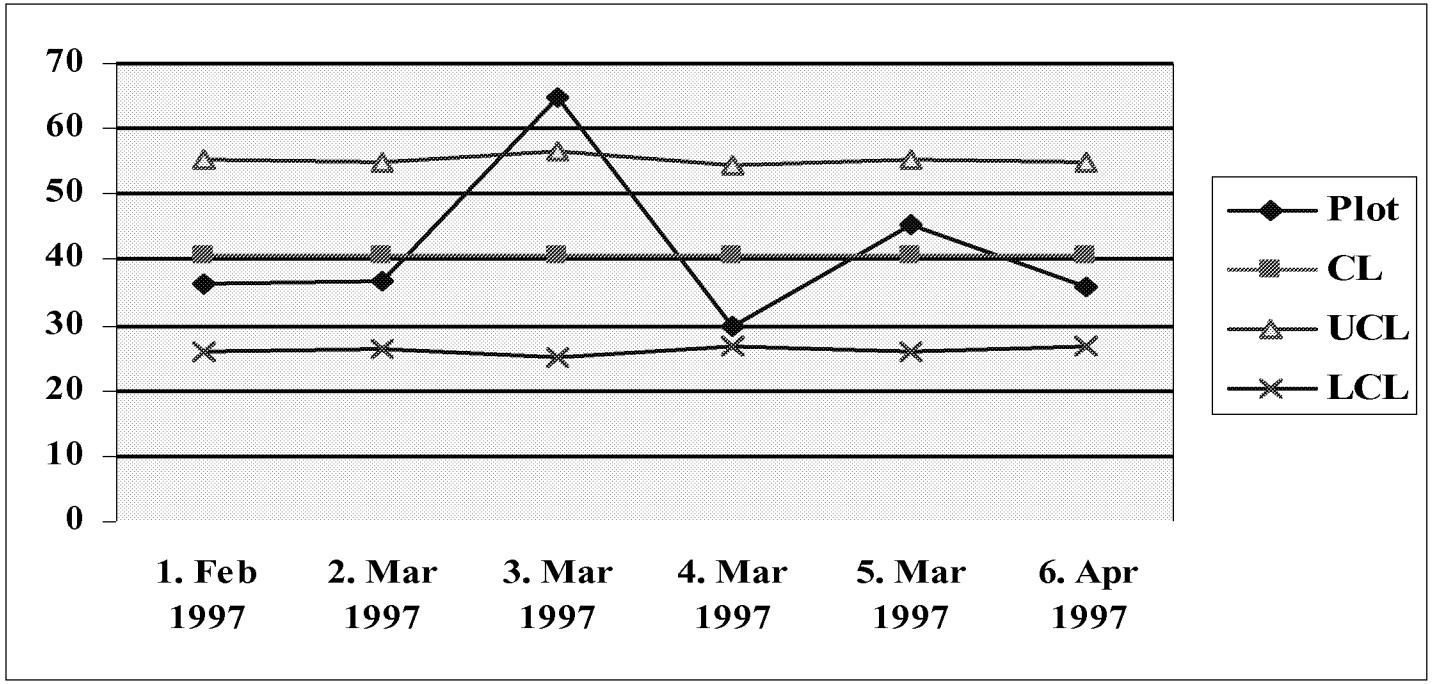
Calculating the limits

- Defects/KSLOC = Number of Defects*1000/SLOC reviewed per sample (calculated for each sample). These are plotted as Plot.
- CL = Total Number of Defects/Total number of SLOC reviewed * 1000
- a(1) = SLOC reviewed/1000 (calculated for each sample)
- UCL = CL+3(SQRT(CL/a(1)) (calculated for each sample)
- LCL = CL-3(SQRT(CL/a(1)) (calculated for each sample)

Calculations for each sample

Sample	Plot	CL	UCL	LCL	a(1)
1. Feb 1997	36.4	40.73	55.4	26.09	1.7
2. Mar 1997	36.7	40.73	55.01	26.45	1.8
3. Mar 1997	65	40.73	56.49	24.97	1.5
4. Mar 1997	29.6	40.73	54.53	26.93	1.9
5. Mar 1997	45.2	40.73	55.47	25.99	1.7
6. Apr 1997	35.8	40.73	54.84	26.63	1.8

Code Peer Reviews Control Chart

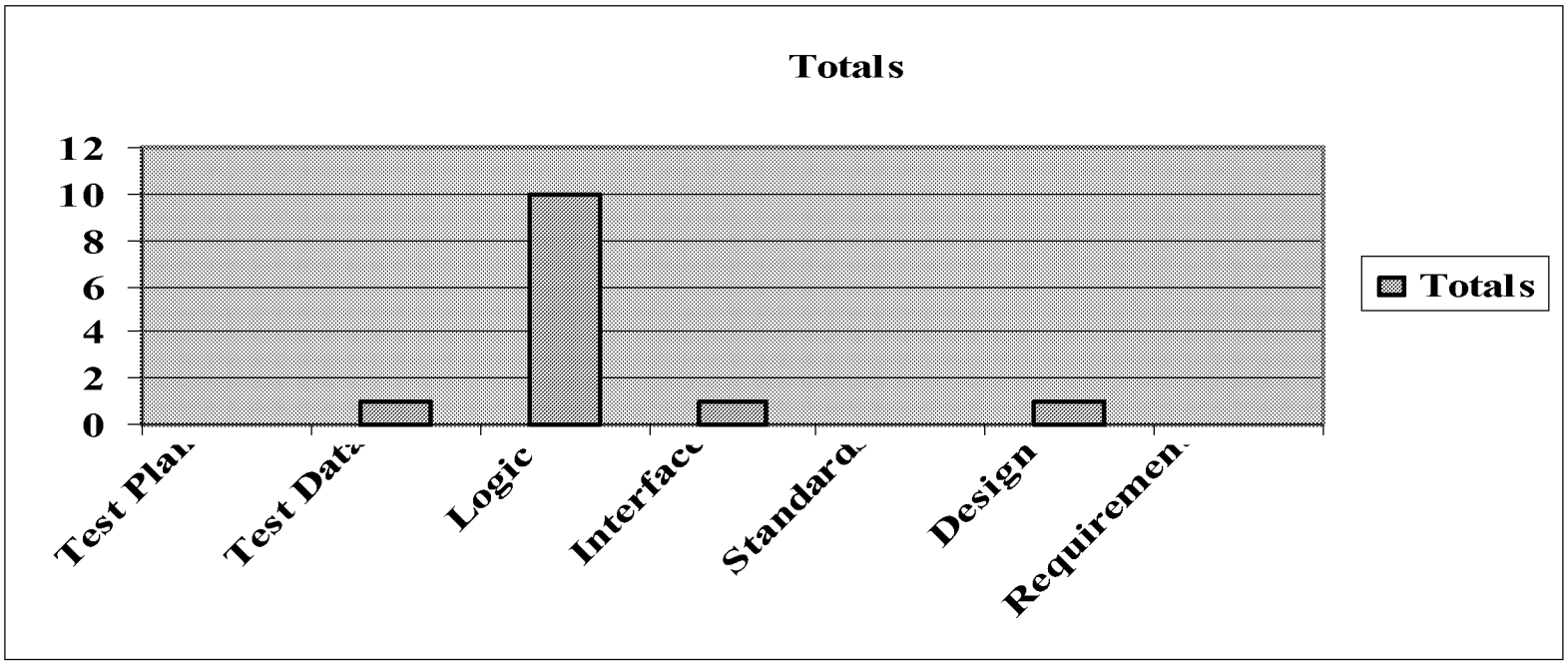


- The process is out of statistical process control in the third sample
- Analysis revealed that this was caused when the project introduced coding standards and many coding violations were introduced

MITRE

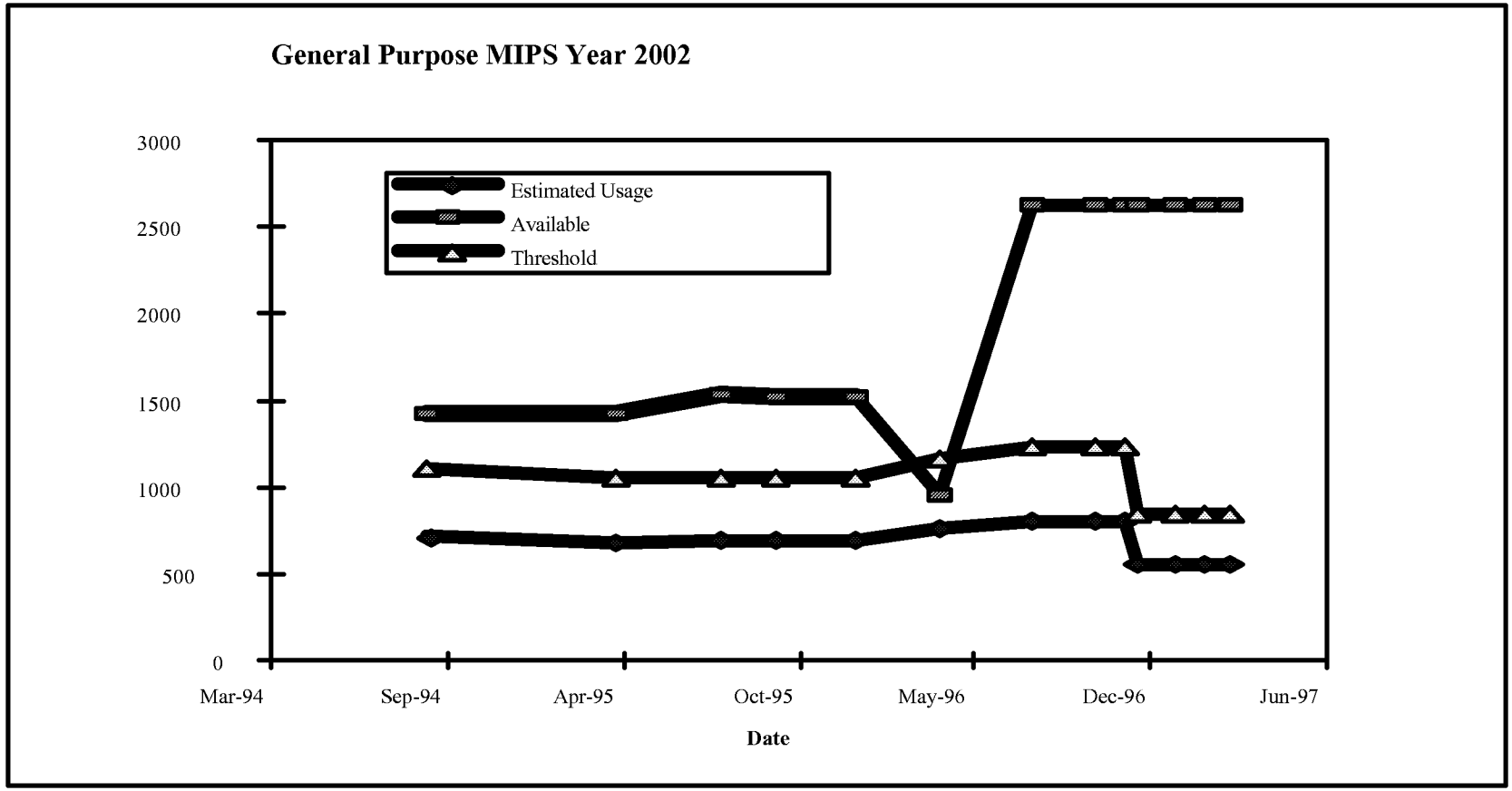
Bar Charts for Thread Tests

Samples	Test Plan	Test Data	Logic	Interface	Standards	Design	Requirements	- Defect Categories
1			6	1		1		
2		1						
3			4					
Totals	0	1	10	1	0	1	0	



You would expect more logic defects than others

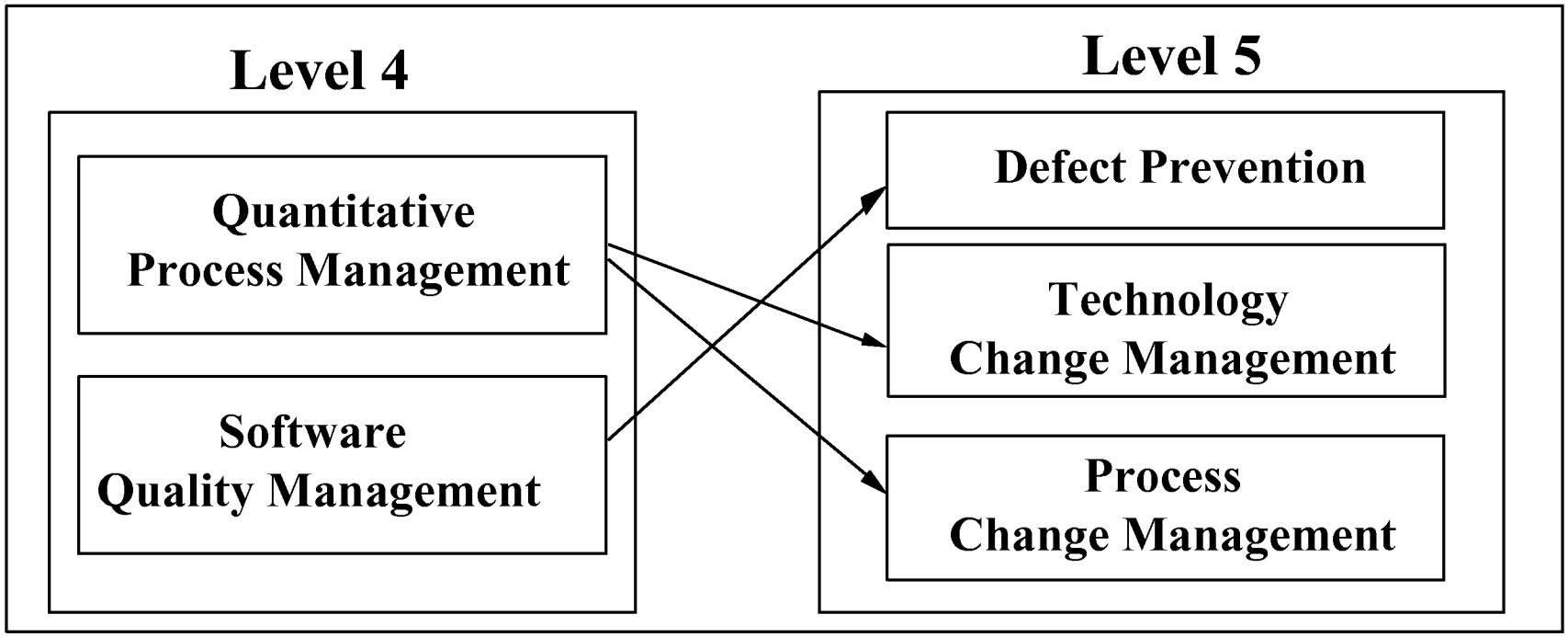
Example
Critical Computer Resources



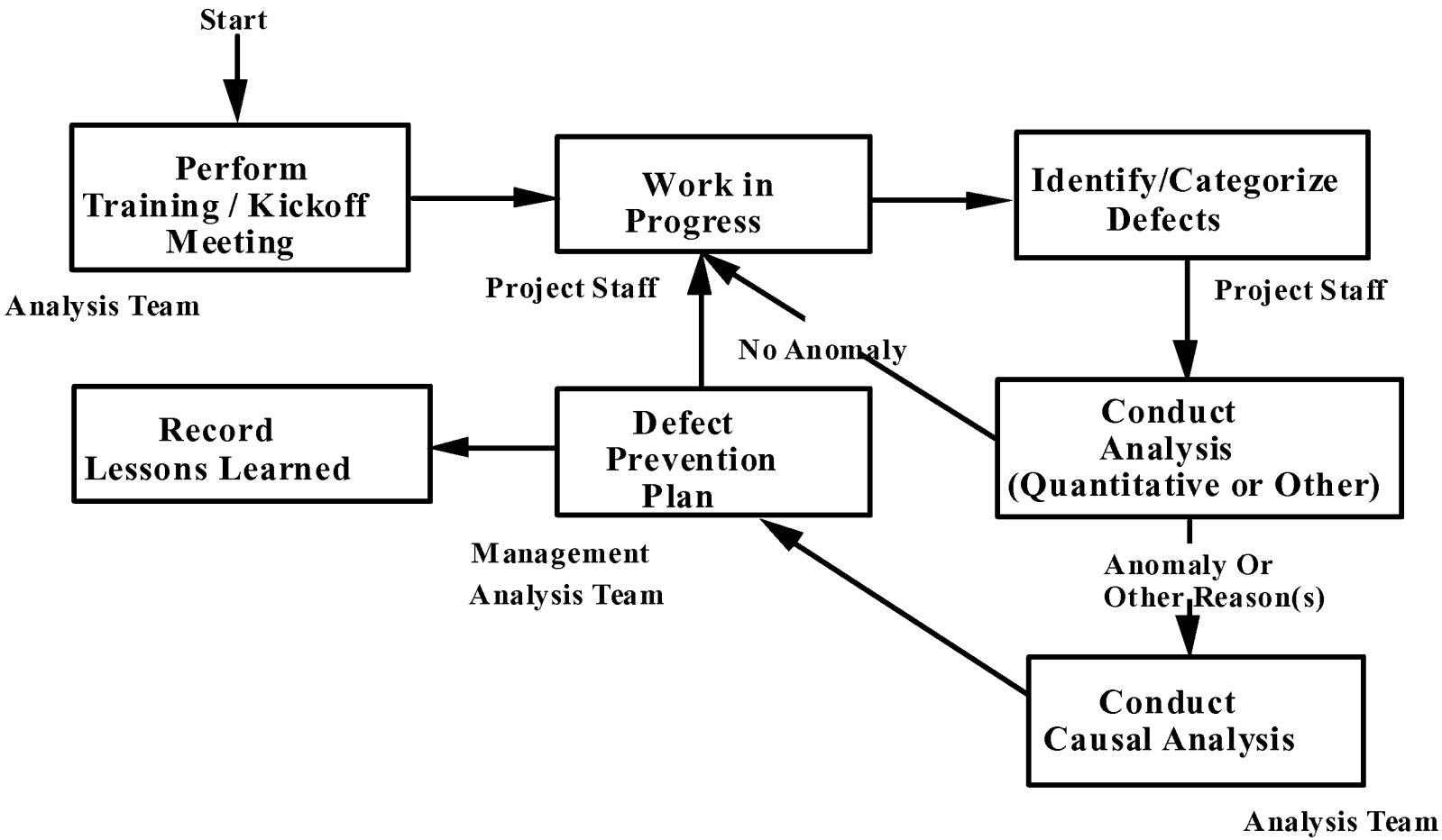
- The customer introduced many new requirements around Nov/Dec 1995
- The model revealed that the MIPS threshold was threatened with increased computations
- More MIPS were added to the architecture in May 1996

Level 4 to Level 5 Relationships

Data analysis at Level 4 enables focusing the on Defect Prevention, Technology Change Management, and Process Change Management at Level 5



Defect Prevention Process



Defect Prevention Plans and Activities

- **Defects can be prevented on a variety of entities:**
  - **Project Plans**
  - **Project Resources**
  - **Quality Goals**
  - **Design**
  - **Interfaces**
  - **Test Procedures**
  - **Processes**
  - **Technologies**
  - **Management**
  - **Project Schedules**
  - **Standards**
  - **Requirements**
  - **Code**
  - **Test Plans**
  - **Documentation**
  - **Procedures**
  - **Training**
  - **Engineering**

Defect Prevention Example

Raw Data - Code Peer Review

Sample	Units	SLOC	Defects	Defects/KSLOC
1. Mar 1998	6	515	15	29.12621359
2. Apr 1998	10	614	16	26.05863192
3. Apr 1998	7	573	7	12.21640489
4. Apr 1998	7	305	7	22.95081967
5. Apr 1998	4	350	21	60
6. Apr 1998	3	205	2	9.756097561
7. Apr 1998	8	701	11	15.69186876
8. May 1998	3	319	3	9.404388715
Totals	76	3582	72	

Defect Prevention Example (Cont.)

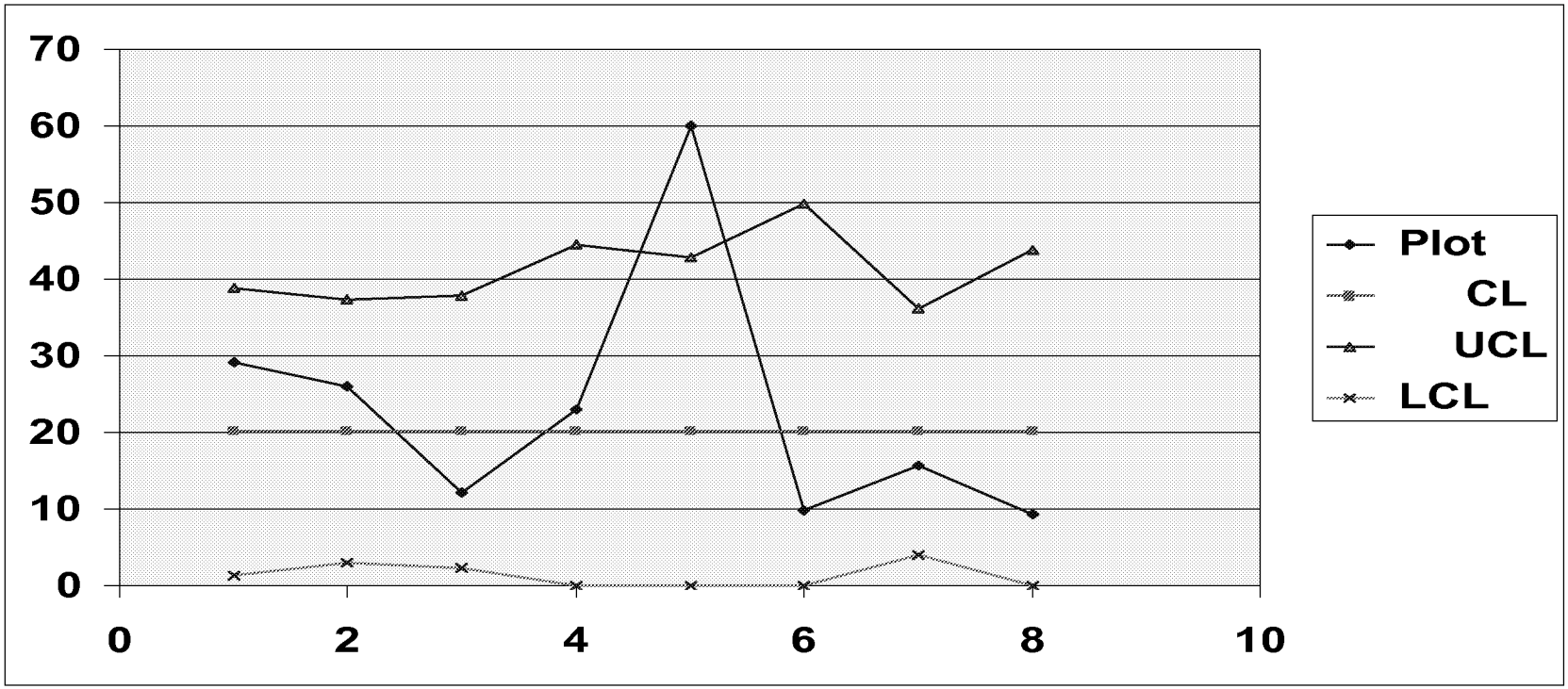
Calculations

Sample	Plot	CL	UCL	LCL	a(1)
1. Mar 1998	29.1	20.1	38.843	1.358279654	0.515
2. Apr 1998	26.1	20.1	37.265	2.935632196	0.614
3. Apr 1998	12.2	20.1	37.869	2.332140203	0.573
4. Apr 1998	23	20.1	44.455	0	0.305
5. Apr 1998	60	20.1	42.835	0	0.35
6. Apr 1998	9.76	20.1	49.807	0	0.205
7. Apr 1998	15.7	20.1	36.165	4.036058356	0.701
8. May 1998	9.4	20.1	43.914	0	0.319

When LCL is negative it is set to zero

Defect Prevention Example (Cont.)

Plot

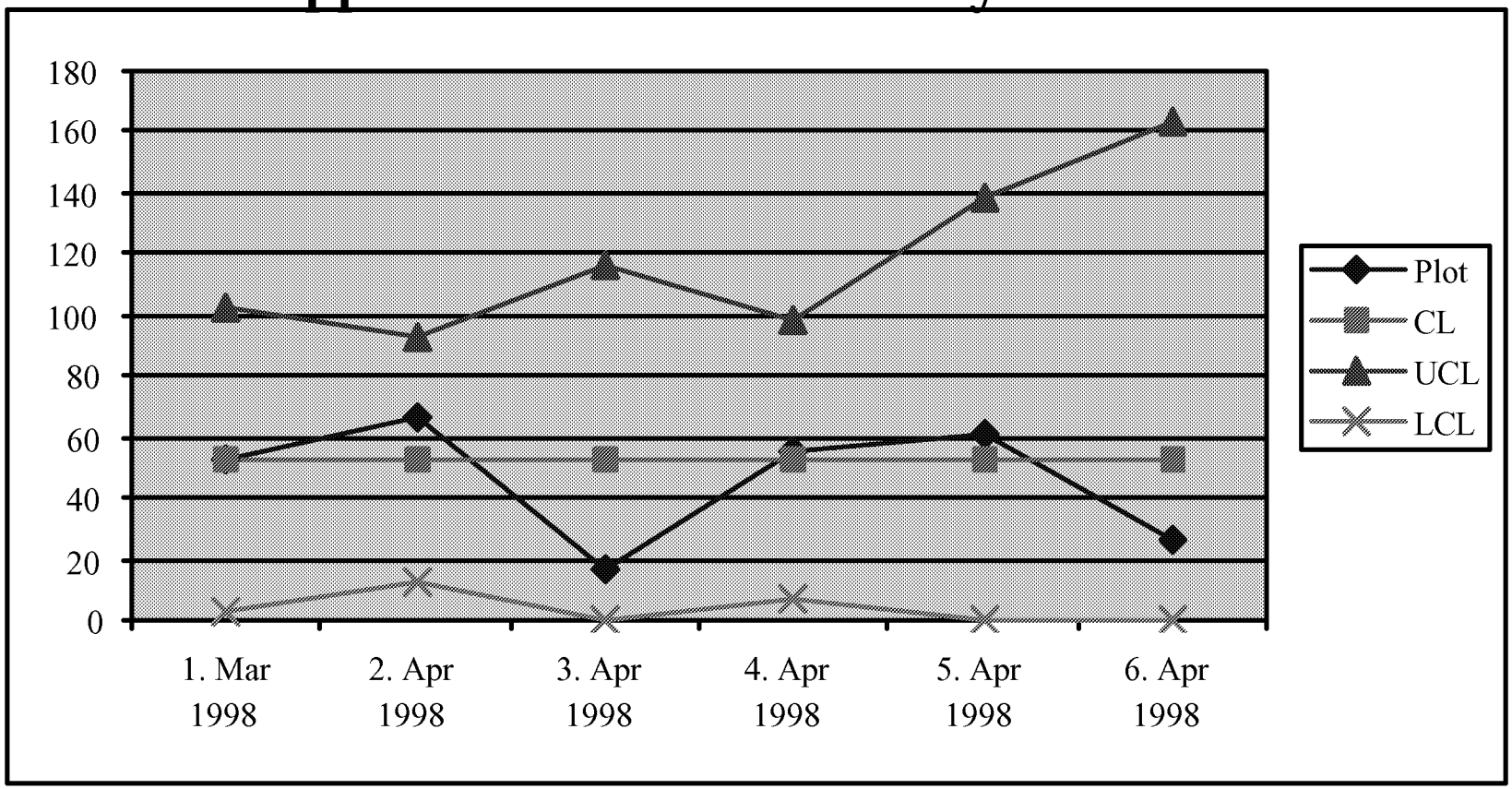


Causal Analysis

- Revealed that data were for database code and applications code
- Control charts require similar data for similar processes
- Apples to apples analogy

Defect Prevention Example (Cont.)

Plot - Applications Code Data Only



Process is now under statistical process control

Defect Prevention Example (Cont.)

Root Cause

- Data gathered from dissimilar activities cannot be used on the same statistical process on control charts
- Data from design cannot be combined with data from coding
- The process for database design and code is different from that used for applications design and code as are the teams and methodologies

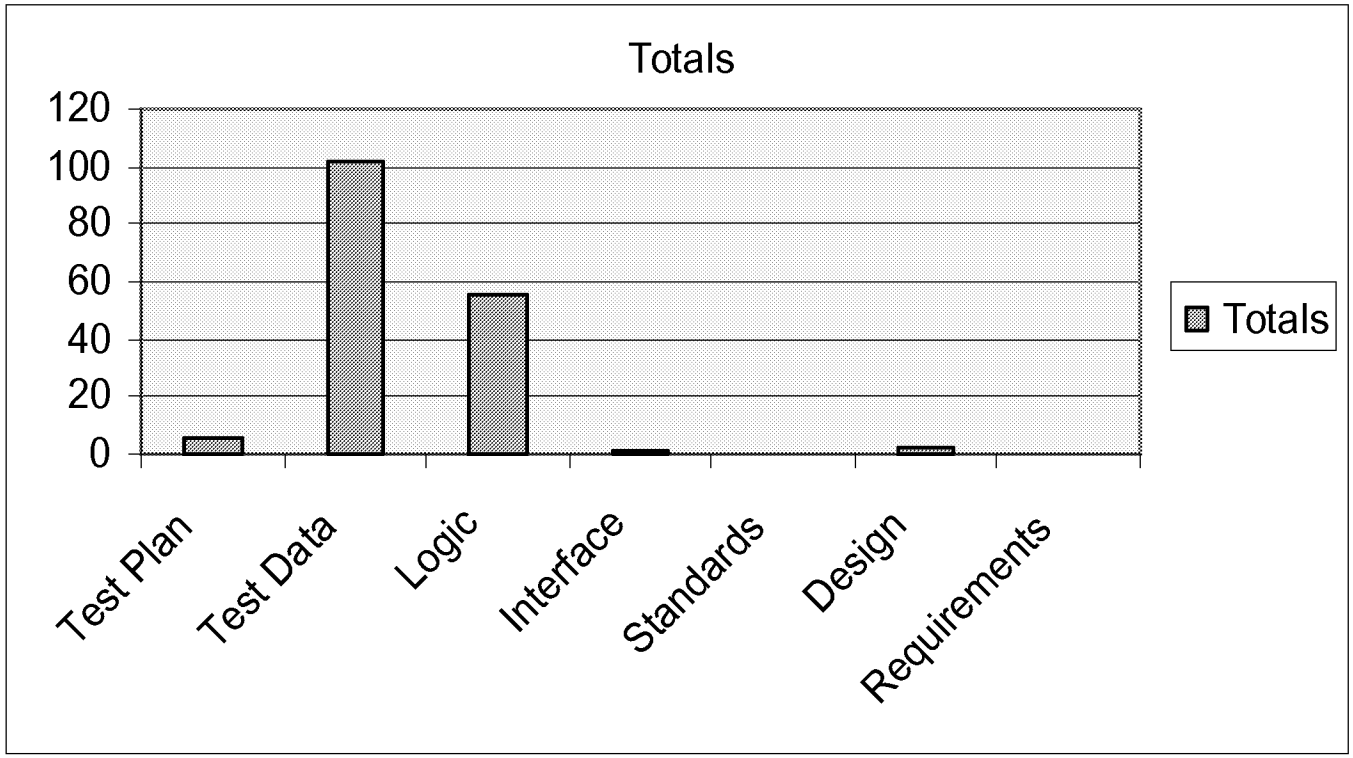
Defect Prevention

The defect prevention is against the process of collecting data for SPC control charts

Thread Tests

Samples	Test Plan	Test Data	Logic	Interface	Standards	Design	Requirements
1	2	6					
2		10					
3	1	9	3				
4	2	1	13				
5		1	7				
6		10	14				
7		4	2				
8		28					
9						2	
10			6				
11	1	1					
12		10					
13		9	1				
14		6	1	1			
15		5	7				
16		2	1				
Totals	6	102	55	1	0	2	0

Bar Chart for Thread Tests



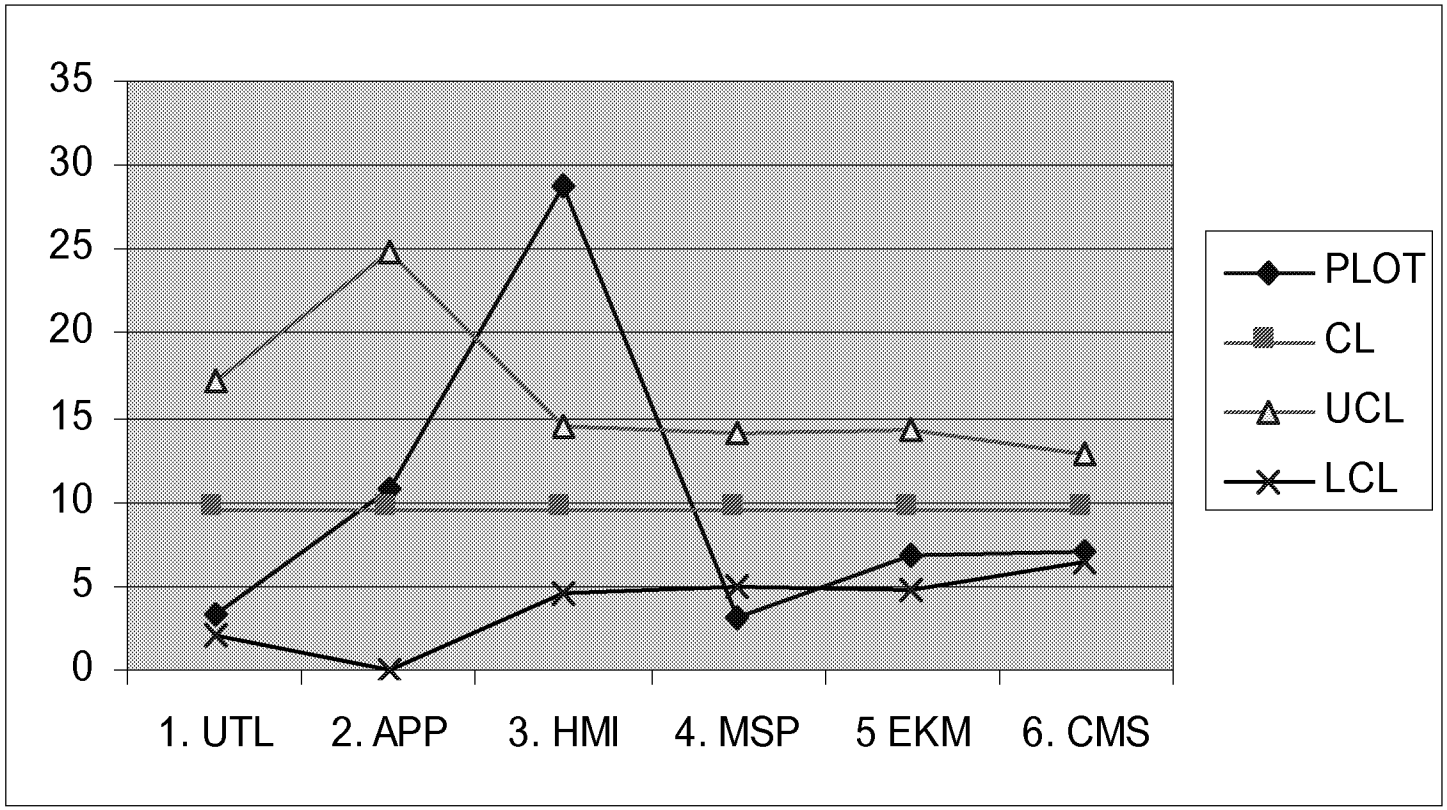
- Test data would not be expected to have the majority of defects
- The root cause is that test procedures had not been peer reviewed
- The defect prevention is to peer review test procedures

Requirements Defects

Sample	SRSs	No. Rqmts	Defects	Defects/100 Rqmts
1. UTL	1	152	5	3.28
2. APP	1	37	4	10.81
3. HMI	1	350	101	28.85
4. MSP	1	421	13	3.08
5. EKM	1	370	25	6.757
6. CMS	1	844	60	7.10
Totals	6	2174	208	

Sample	PLOT	CL	UCL	LCL	a(1)
1. UTL	3.28	9.56	17.09	2.04	1.52
2. APP	10.81	9.56	24.82	0	0.37
3. HMI	28.85	9.56	14.52	4.60	3.5
4. MSP	3.08	9.563	14.09	5.04	4.21
5. EKM	6.75	9.563	14.39	4.74	3.7
6. CMS	7.10	9.56	12.76	6.37	8.44

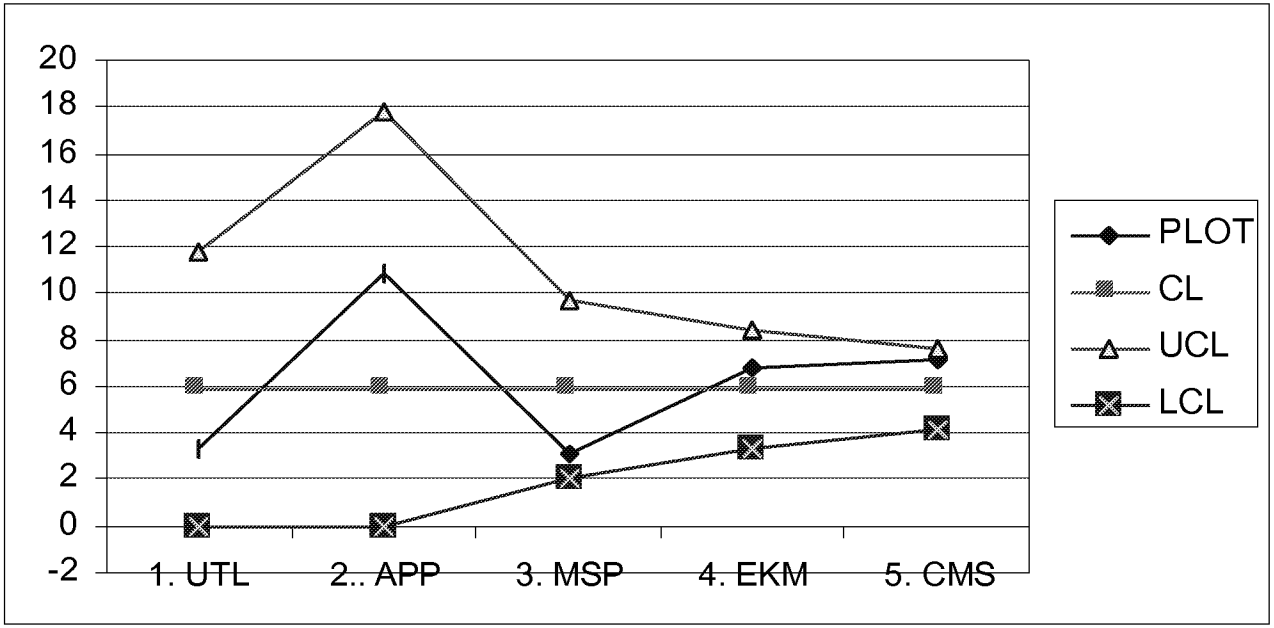
Requirements Defects (Cont.)



- HMI is Human Machine Interface, Others are Applications
- Again, dissimilar activities cannot be used on the same statistical process on control charts

Requirements Defects (Cont.)

Sample	SRSs	No. Rqmts	Defects	Defects/100 Rqmts
1. UTL	1	152	5	3.29
2. APP	1	37	4	10.81
3. MSP	1	421	13	3.09
4. EKM	1	370	25	6.76
5. CMS	1	844	60	7.11
Totals	5	1824	107	



Without HMI

References

- **Software Engineering Institute *Capability Maturity Model* (SEI CMM) V1.1,** 1993
- ***Software Measures and the Capability Maturity Model*, John H. Baumert, Mark S. McWhinney, SEI, 1992**
- ***Practical Software Measurement: Measuring for Process Management and Improvement*, William A. Florac, Robert E. Park, Anita D. Carleton, SEI, April, 1997**
- ***An SPC Primer*, Quality America, Inc., 1984, Thomas Pyzdek**
- ***Getting to Level 4 in the CMM*, Ron Radice, SEPG 97, San Jose, CA, 1997**
- ***Statistical Process Control for Software*, Anita Carleton, Mark C. Paulk, SEI, The 97 Software Engineering Symposium, 1997**
- ***Understanding Statistical Process Control*, David S. Chambers & Donald J. Wheeler, SPC Press, 1995**
- ***Juran's Quality Control Handbook*, 4th Edition, McGraw-Hill Book Company, 1988**
- ***Understanding Variation, The Key to Managing Chaos*, Donald J. Wheeler, SPC Press, 1993**
- ***Advanced Topics in Statistical Process Control*, Donald J. Wheeler, SPC Press, 1995**
- ***On Probability As a Basis For Action*, The American Statistician, Vol. 29, No.4 (146-152), Edwards W. Deming, November 1975**
- ***Assumptions for Statistical Inference*, The American Statistician, Vol. 47, No.1 (1-11), Gerald J. Hahn & William Q. Meeker, February 1993**
- ***Managing the Software Process*, Watts S. Humphrey, SEI Series in Software Engineering, Addison-Wesley Publishing Company, September 1997**

Contact Information

Alfred (Al) W. Florence

Florence@mitre.org

(703) 883-7476

Empirical Study of Inspection and Testing Data at Ericsson, Norway

Reidar Conradi, Norw. Univ. of Technology and Science (NTNU), Trondheim, Norway #
Amarjit Singh Marjara, Cap Gemini AS, Trondheim, Norway ##
Børge Skåtevik, STC, Vatlandsvåg, Norway

p.t. Univ. Maryland, Phone +1 (301)405-1255, Fax –6638, conradi@idi.ntnu.no,
Phone +47 73.829111, amarjit.marjara@capgemini.no.

This paper was presented at PROFES'99, Oulu, Finland, 22-24 June 1999.
This is a revised version for 24th NASA Software Eng. Workshop, Washington, 1-2 Dec. 1999.

Abstract

Inspections and testing represent core techniques to ensure reliable software. Inspections also seem to have a positive effect on predictability, total costs and delivery time.

This paper presents a case study of inspections and testing, done at the Ericsson development department outside Oslo in Norway. This department develops and maintains customer-defined services around AXE phone switches, i.e. the functionality around the "star"" and "square" buttons on house telephones.

AXE development at Ericsson world-wide uses a simple, local experience database to record inspections and testing data. Two MSc students from NTNU have been given access to such historical data in 1997 [Marjara97] and 1998 [Skaatevik99]. The results from these two diploma theses constitute the basis for this paper.

The paper will study questions such as:

- The effectiveness and cost-effectiveness of inspections,
- The cost-effectiveness and defect profile of inspection meetings vs. individual reading,
- The relation between complexity/modification-rate and defect density,
- Whether the defect density for modules can be predicted from inspections for later phases and deliveries.

The paper is organized as follows: Section 1 summarizes some relevant parts of the state of the art, especially of inspections. Section 2 first describes the Ericsson context, and Section 3 describes questions and hypotheses for the study. Section 4 describes the organization of the study, and Section 5 presents and discusses the results. Section 6 sums up the paper and recommends some future work.

Contents

Preface

The paper will present results from two MSc theses at NTNU, that have analyzed historical defect data at Ericsson in Oslo, Norway -- related to their AXE switches. Ericsson has practised Gilb inspections for many years, and collects defect data from inspections and testing in a small database.

These studies revealed that inspections indeed are the most cost-effective verification technique. Inspections tend to catch 2/3 of the defects before testing, by spending 10% of the development effort and thereby saving about 20% of the effort (by earlier defect correction, a ``win-win''). Inspection meetings were also cost-effective over most testing techniques, so they should not be omitted. Inspection meetings also found the same type of errors (Major, Super Major) as individual inspections.

We also found that there is a correlation between module complexity, modification rate, and the defect density found during field-use, but not during inspections and test. Due to missing data, we

could not find out whether the defect density of modules repeated itself across inspection/test phases and over several deliveries, i.e. we could not predict ``defect-prone" modules. Defect classification was also unsatisfactory, and prevented analysis of many interesting hypotheses.

1. State of the art

Quality in terms of reliability is of crucial importance for most software systems.

Common remedies are sound methods for system architecture and implementation, high-level languages, formal methods and analysis, and inspection and testing techniques. Especially the latter two have been extensively described in the literature, and vast empirical materials have been collected, analyzed and published. This paper only refers to general test methods, so we will not comment on these here.

Inspections were systematized by Fagan [Fagan76] [Fagan86] and represent one of the most important quality assurance techniques. Inspections prescribe a simple and well-defined process, involving group work, and have a well-defined metrics. They normally produce a high success rate, i.e. by spending 10% of the development effort, we diagnose 2/3 of the defects before testing, and save 20% of the total effort -- a win-win: so "quality is free". Inspections can be applied on most documents, even requirements [Basili96]. They also promote team learning, and provide a general assessment of reviewed documents.

Of current research topics are:

- The role of the final inspection meeting (emphasized by Tom Gilb [Gilb93], see also [Votta93].
- When to stop inspections?
- When to stop testing, cf. [Adams84]?
- The effect of root-cause-analysis on defects.
- The role of inspection vs. testing in finding defects, e.g. their relative effectiveness and cost-effectiveness.
- The relationship between general document properties and defects.
- Defect densities of individual modules through phases and deliveries.

Our research questions and hypotheses deal with the three latter.

2. The company context

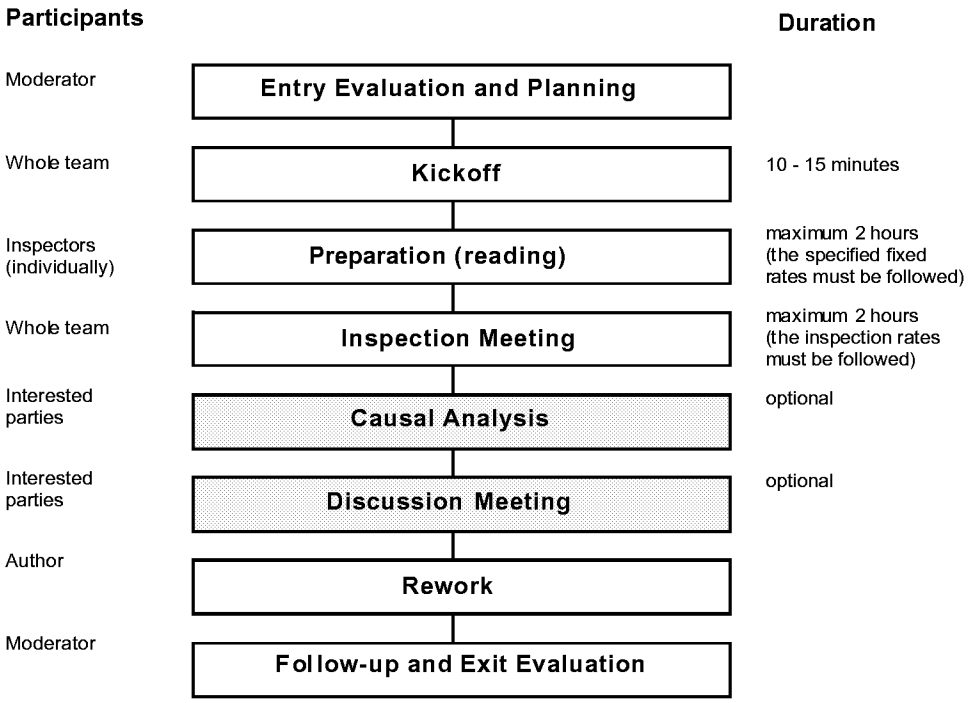
Ericsson employs about 100,000 people world-wide, whereof 20,000 in development. They have company-wide and standardized processes for most kind of software development, with adaptations for the kind of work being done. Ericsson has adopted a classical waterfall model, with so-called "tollgates" at critical decision points. In all this, verification techniques like inspections and testing are crucial. Inspection is done for every life-cycle document, although we will mostly look at design and code artifacts. Testing consists of unit test, function test and system test, where the two latter may be done at some integration site different from the development site (e.g. Stockholm).

We will only study design inspections (in-groups), simplified code reviews (by individuals) and partly testing in this paper.

The inspection process at Ericsson is based on techniques originally developed by Michael Fagan [Fagan76] at IBM and refined by Tom Gilb [Gilb93]. The process is tailor-made by the local development department. In addition there is a simplified code review done by individual developers (data from code review and unit test are sometimes merged into a "desk check"). Thus full inspections are only done upon design documents in our studies. Data from inspections/reviews and testing are collected in a simple, proprietary database and used for local tuning of the process. Defects are classified in Major, SuperMajor and Questions (the latter is omitted here) -- thus no deep categorization.

We have studied software development at the Ericsson site outside Oslo. It just passed CMM level 2 certification in Oct. 1998, and aims for level 3 in year 2000. The Oslo development site has about 400 developers, mostly working on software. The actual department has about 50 developers, and works mostly on the AXE-10 digital software switch, which contains many subsystems. Each subsystem may contain a number of modules. The development technology is SDL design language (SDT tool from Telelogic) and their proprietary PLEX language from the late 1970s (own compilers and debuggers).

Figure 1. Basic inspection process at Ericsson for design artifacts (documents).



The first level inspection process

Special inspection groups are formed, called product committees (PC), to take care of all impacts on one subsystem. In this paper, we will only look at subsystem-internal inspections, not across subsystems. The inspection process is indicated in figure 1 above, and follows Fagan/Gilb inspections wrt. overall set-up, duration etc. The number of inspectors per document is typically 3-4. Special check-lists are used for each document type.

The different types of documents are presented in the table 1 below.

Table 1. Document types (18 such).

Document type	Application Information
ADI	Adaptation Direction
AI	Application Information
BD	Block Description
BDFC	Block Description Flow Chart
COD	Command Description
FD	Function Description
FDFC	Function Description Flow Chart
FF	Function Framework
FS	Function Specification
FTI	Function Test Instruction
FTS	Function Test Specification
IP	Implementation Proposal
OPI	Operational Instruction
POD	Printout Description
PRI	Product Revision Information
SD	Signal Description
SPL	Source Parameter List
SPI	Source Program Information

Each of these document types have specific, recommended inspection rates (Skåtevik99).

3. Questions and hypotheses

3.1 One Observation

O1: How (cost-)effective are inspections and testing?

3.2 Three Questions

Q1: Are inspections performed at the recommended inspection rates?

Q2: How cost-efficient are the inspection meetings?

Q3: Are the same kind of defects found in initial inspection reading and following inspection meetings?

3.3 Three Hypotheses

For each question we present one null hypothesis, $\mathbf{H}_0$, which is the one that will actually be tested, and an alternative hypothesis, $\mathbf{H}_a$, which may be considered valid if the null hypothesis is rejected. For the statistical tests presented in this paper, a significance level (p-level) of 0.10 is assumed.

The three alternative hypotheses are:

H1: Is there a significant, positive correlation between defects found during field-use and document complexity?

H2: Is there a significant, positive correlation between defects found during inspection/test and document complexity?

H3: Is there a significant correlation between defect rates across phases and deliveries for individual documents/modules? (i.e. try to track "defect-prone" modules)?

4. Organization of the study

We have performed two studies where we have collected and analyzed historical data from software department at Ericsson in Oslo. Torbjørn Frotveit, our middleman at Ericsson, has all the time furnished us with the requested data.

This paper presents results from these two studies of inspection and testing:

- **Study 1:** This is the work done in a diploma thesis from 1997 [Marjara97]. Marjara investigated inspection and test data from Project A of 20,000 person-hours (14 person-years). Defect data in this work included inspection, desk check, function test, system test and partly field-use.

- **Study 2:** This is the follow-up work done in the diploma thesis from 1998 [Skåtevik99]. This thesis has data from 6 different projects (Project A-F), including the project Marjara used in Study 1. It represents over 100,000 person-hours (70 person-years). The test data in this work include only data from inspection and desk check, since later testings were done by other Ericsson divisions. However, it was possible to split desk check in code review and unit test, and data from these to activities are presented. Data from field-use are not included, due to same reasons as for function- and system test.

Threats to internal validity:

We have used standard indicators from the literature on most properties (defect densities, inspection rates, effort consumption etc.), so all in all we are on agreed ground. However, wrt. Module complexity we are unsure, and further studies are needed. Whether the recorded defect data in the Ericsson database are trustworthy is hard to say. We certainly have discovered inconsistencies and missing data, but our confidence is pretty high.

Threats to external validity:

Since Ericsson has standard working processes world-wide, we can assume at least company-wide relevance. However, many of the findings are also in line with previous empirical studies, so we feel confident on general level.

5. The results and evaluation of these

This chapter presents the results from the two studies described in the previous section (4), and tries to conclude the questions and hypotheses stated in section 3.
Two definitions will be used throughout this section, effectiveness and cost-effectiveness:

Effectiveness: the degree to which a certain technique manages to find defects, i.e. diagnosed defect rate (defects per "volume-unit"), regardless of cost. This is sometimes called efficacy.

Cost-effectiveness: effort spent to find one defect.

5.1 O1: How (cost-)effective are inspections and testing?

Here we shall describe and compare the effectiveness and cost-effectiveness of inspections and testing at Ericsson in Oslo. The effort spent *before* invidual reading is proportionally distributed over inspection reading and inspection meetings. The inspection-phase effort spent *after* inspection meetings are similarly merged into "defect fixing" (se Figure 1). Table 2 is taken from Study 1 and shows the effectiveness of inspections and testing. All efforts are in person-hours, sometimes just called hours.

Table 2. Efficiency: total defects found, Study 1.

Activity	Defects [#]	[%]
Inspection reading, design	928	61.8
Inspection meeting, design	29	1.9
Desk check (code review + unit test)	404	26.9
Function test	89	5.9
System test	17	1.1
Field-use	35	2.3
Total	1502	100.0

Table 2 shows that inspections are the most effective verification activity, finding almost 64% of total defects found in the project. Second best is the desk check that finds almost 27%. We also see that 3% of the defects found by inspections are found in the meetings. To analyze which of the verification activities that are most effective, the effort spent on the different activities was gathered. Table 3 shows the effort (person-hours) spent on the six verification activities.

Table 3. Effort and cost-efficiency on inspection and testing, Study 1.

Activity	Defects [#]	Total effort on defect detection [h]	Cost-effectiveness [h:m per defect]	Total effort on defect fixing [h]	Estimated saved effort by early defect removal ("magic formulae") [h]
Inspection reading, design	928	786.8	00:51	311.2	8200
Inspection meeting, design	29	375.7	12:57		
Code review and unit test	404	1257.0	03:07	-	-
Function test	89	7000.0	78:39	-	-
System test	17	-	-	-	
Field-use	35	-	-	-	

When combining effort and number of defects, inspections proved to be the most cost-effective. Not surprisingly, function test is the most expensive activity (note: we have no effort data om system test). It should be noted that only human labor is included for desk check (code review and unit test) and function test. The costs of computer hours or special test tools are not included. Neither is the human effort spent in designing the test cases.

In Study 2 it was not possible to get defect data from function test, system test and field-use (representing 9.3% of the defects in Study 1). Instead the data made it possible to split up the desk check, which actually consist of code review and unit test (emulator test). Table 4 shows the results.

Table 4. Efficiency: total defects found, Study 2.

Activity	Defects [#]	[%]
Inspection reading, design	4478	71.1
Inspection reading, design	392	6.2
Desk check, code	832	13.2
Unit test, code	598	9.5
Total	6300	100.0

Again, the data show that inspections are highly effective, contributing to 77% of all the defects found in the projects. Desk check is second best, finding almost 13% of the defects in the projects. Compared to Study 1, there is an improvement in the inspection meeting, whose effectiveness has increased from 3% to 8% for defects found during inspections.

Table 5 shows the effort (person-hours) of the different activities from Study 2. In this study, no data from Function test or later tests were available.

Table 5. Effort and cost-efficiency on inspection and testing, Study 2.

Activity	Defects [#]	Total effort on defect detection [h]	Cost-effectiveness [h:m per defect]	Total effort on defect fixing [h]	Estimated saved effort by early defect removal ("magic formulae") [h]
Inspection reading, design	4478	5563	01:15	11737	41000
Inspection meeting, design	392	3215	08:12		
Desk check, code	832	2440	02:56		-
Unit test, code	598	4388	07:20		-

The inspection meeting itself is more cost-effective in Study 2 (8h:12min per defect) than in Study 1 (12h:57min per defect).

In Study 2 covering 100,000 person-hours, a total of 20,515 person-hours were spent on inspections (including 11,737 person-hours on defect fixing). It has been calculated that inspections did save 41,000 person-hours, which would have been necessary to locate and correct defects otherwise found by later testing. That is, a net saving of 21% of the total project effort.

Study 1 covered 20,000 person-hours where 1474 person-hours were spent on inspections (including 311.2 person-hours on defect fixing). In this study it was calculated that Ericsson saved 8200 person-hours, or a net saving of 34%!

5.2 Q1: Are inspections performed at the recommended inspection rates?

Here we want to see if the recommended inspection rates were actually applied. The results are presented in table 6. Note, that all this applies to design documents, not source code.

Table 6. Recommened rate versus actual total effort during inspections in Study 2.

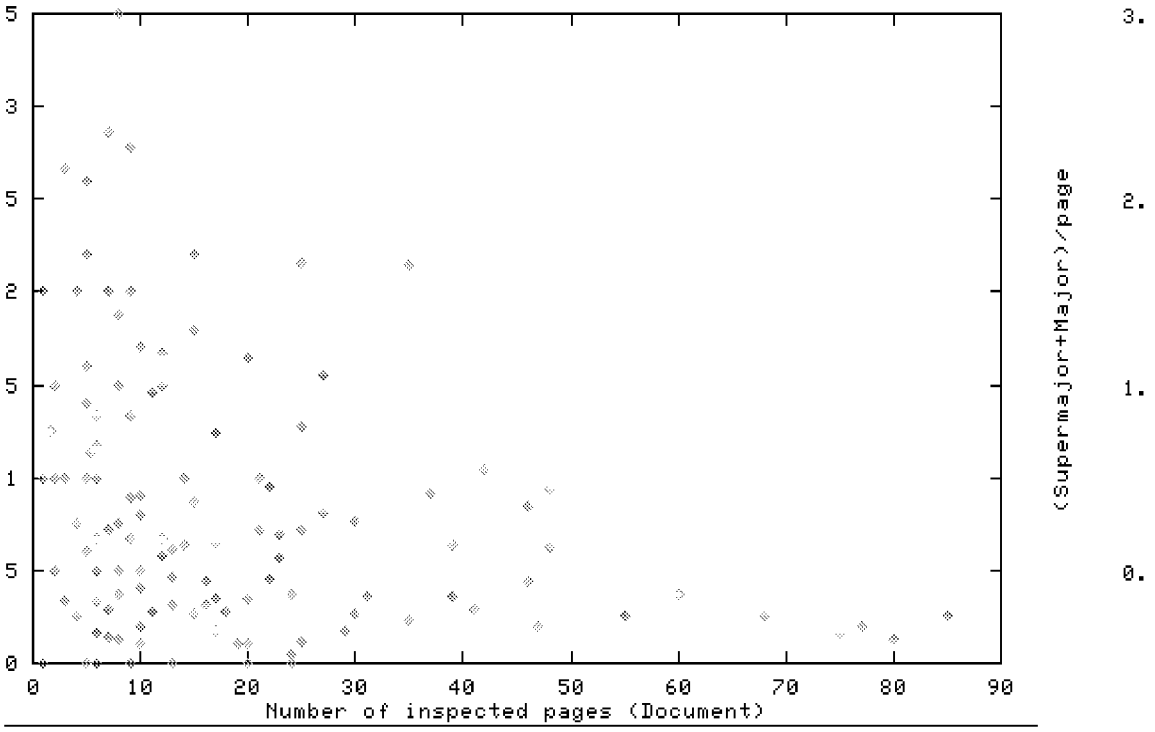
Type of effort	Total inspection effort including defect fixing [h]	Share [%]
Actual effort, Study 1	1474	54%
Recommended inspection rate, Study 1	2723	--
Actual effort, Study 2	20,515	78,6%
Recommended inspection rate, Study 2	26,405	--

Thus in Study 2, inspections are performed too fast. Only 20,515 person-hours are actually spent on inspections including defect fixing – being 78.6% of the recommended expediture of 26,405 person-hours. The average number of defects per page is 0.43.

Study 1 concluded with even more deviating results, as only 54% (1474 actual person-hours out of 2723 recommended person-hours) are totally used during inspections including defect fixing.

As reported elsewhere, plots on reading rate and defect detection rate (see figure 2) show that the number of defects found per page decreases as the number of inspected pages (document length) per hour increases. Inspection performed too fast will then result in decreased detection rate. However, we have not done any (re)analysis of "optimal" reading rates here. Also note, that the individual reading rate is a *part* of the total inspection rate mentioned e.g. in Table 6.

Figure 2. Number of pages inspected and defect detection rate, Study 1.



5.3 Q2: How cost-efficient are the inspection meetings?

Table 7 shows the effort consumption for each step of the inspections including defect fixing from Study 2. Effort *before* individual reading and inspection meeting has been proportionally distributed on these two activities.

Table 7. Effort consumption for inspection and defect fixing, Study 2.

	Inspection Reading	Inspection Meeting	Defect fixing	Sum
Person-hours	5563	3215	11737	20515
[%]	27.12%	15.67%	57.21%	100.00%

Note that 57.2% of the "inspection-time effort" is spent on defect fixing in Study 2 (11,737 of 20,515 person-hours), while only 21.1% is spent on such (311.2 out of 1473.7 person-hours) in Study 1.

Table 8 from Study 2, shows the number of defects recorded in reading, in meetings, and the total.

Table 8. Cost-effectiveness and defect classification from inspections, Study 2.

	Major defects		Super Major defects		Sum defects	Defect detection effort	Cost-effectiveness
	[#]	[%]	[#]	[%]	[#]	[h]	[h:m per defect]
Inspection Reading	4356	97.2%	122	2.7%	4478	5563	01:15
Inspection Meeting	380	96.9%	12	3.1%	392	3215	08:12
Entire inspection	4736	97.2%	134	2.7%	4870	8778	01:48

As mentioned, the defects are classified in two categories:

- **Major:** Defects that can have a major impact later, that might cause defects in the end products, and that will be expensive to clean up later.

- **Super Major:** Defects that have major impact on total cost of the project.

In Study 2, 8% of the defects found by inspections are found in the meetings, with a cost-effectiveness of 8h:12min of person-effort. Compared to function test and system test, inspection meetings are indeed cost-effective in defect removal.

5.4 Q3: Are the same kind of defects found in initial inspection reading and following inspection meetings?

We will also like to investigate what type of defects are found during inspection reading versus inspection meetings. Note: We do not have data on whether inspection meetings can refute defects reported from individual reading ("false positives"), cf. [Votta93]. Our data only report new defects from inspection meetings ("true negatives"). Table 8 from Study 2 shows, that totally 2.7% of all defects from inspections are of type Super Major, while the rest are Major.

For inspection reading, the Super Major share is 2.7%. For inspection meeting the share is 3.1%, i.e. only slightly higher. We therefore conclude that inspection meetings find the same "types" of defects as by individual reading.

No such data were available in Study 1.

5.5 H1: Correlation between defects found during field-use and document complexity

Intuitively, we would say that defects detected in field-use could be related to complexity of the module, and to the modification rate for the module. The modification rate indicates how much the module is changed from the base product, and the complexity is represented by the number of states per module (taken from a state machine diagram and reported by TeleLogic's SDL tool called SDT). For new modules the modification grade is zero. Correlation between modules and defect rates for each module (i.e., not the absolutely number of defects, but defects per volume-unit) have not yet been properly checked.

In Study 1, the regression equation can be written as:

$$N_{fu} = \alpha + \beta N_s + \lambda N_{mg}$$

where N_{fu} is number of defects (faults) in field-use, N_s is number of states, N_{mg} is the modification grade, and α, β, and λ are coefficients. $\mathbf{H}_1$ can only be accepted if β and λ are significantly different from zero and the significance level for each of the coefficients is better than 0.10. The following values were estimated:

$$N_{fu} = -1.73 + 0.084 * N_s + 0.097 * N_{mg}$$

Predictor	Coefficient	StDev	t	P
Constant	-1.732	1.067	-1.62	0.166
States	0.084	0.035	2.38	0.063
Modrate	0.097	0.034	2.89	0.034

Here are s = 1.200, R^2 = 79.9%, and $R^2_{(adj)}$ = 71.9%, where s is the estimated standard deviation about the regression line, R^2 is the coefficient of determination, and $R^2_{(adj)}$ is similar but adjusted for degrees of freedom. That is, if a variable is added to an equation, R^2 will get larger, even if the added variable is of no real value. To compensate for this, $R^2_{(adj)}$ is chosen as coefficient of determination.

The values for estimated coefficients are given above, along with their standard deviation, *t*-value for testing if the coefficient is 0, and *p*-value for this test. The analysis of variance is summarised below:

Source	DF	SS	MS	F	P
Regression	2	28.68	14.34	9.96	0.018
Error	5	7.20	1.44		
Total	7	35.88			

In the table above, DF is the degrees of freedom, SS is the total sum of squares corrected for the mean, MS is mean sum of squares, F is the Fisher observator for F-test, and P is the significance level for this test.

It should be noted that the coefficients are not significant, but that the states and modification rate are significant. The F-Fisher test is also significant, and therefore the hypothesis $\mathbf{H}_1$ *can be accepted,* based on the results from the regression analysis.

5.6 H2: Correlation between defects found during inspection/test and document complexity

The relevant data come from Study 2. Because just some of the modules are found over several lifecycles, only 12 modules out of 443 could be used for this analysis. 12 modules out of 443, shows that we should probably have checked out more thoroughly relations between phases in same lifecycle, not just between different lifecycles.

Since data are collected for each document type, and each module in each phase consists of different number of document types, one document type is selected through all the phases. The document type selected is BDFC (Block Description Flow Chart). Table 9 shows the results. Field marked with "-" means that the data are missing, or no module exists. Because all the modules presented in this table only were included in project A through E, project F were excluded.

Table 9. Defect data for BDFC documents over different modules and projects, Study 2.

Module name	Project A			Project B			Project C			Project D			Project E		
	Def/page	Complexity	Defect found basic test	Def/page	Complexity	Defect found basic test	Def/page	Complexity	Defect found basic test	Def/page	Complexity	Defect found basic test	Def/page	Complexity	Defect found basic test
SUSAACA	0.04	72.0	25	0.28	80.5	3	-	-	-	-	-	-	-	-	-
SUSAACT	0.10	177.5	12	0.10	179.0	4	-	-	-	-	-	-	-	-	-
SUSCCTB	0.42	117.5	58	0.80	120.5	24	-	-	-	-	-	-	-	-	-
SUSCR	-	-	-	0.13	95.5	11	-	-	-	3.80	89.00	-	-	-	-
SUSCWC	0.29	-	23	0.10	-	-	-	-	-	-	-	-	-	-	-
SUSCWHF	-	-	11	0.50	-	-	-	-	-	-	-	-	-	-	-
SUSCWP	0.06	220.5	7	0.27	240.0	13	-	-	-	-	-	-	-	-	-
SUSSCR	0.08	244.5	22	-	295.5	34	-	-	-	-	-	-		-	-
SUSACF	0.14	47.0	32	0.37	62.5	28	-	-	-	-	-	-	0.24	66.0	-
SUSAP	0.26	67.0	42	-	-	10	-	-	-	-	-	-	0.04	78.0	-
SUSCCTA	0.34	269.5	118	-	297.5	132	1.00	299.5	3	-	-	-	-	-	-
SUSCS	0.06	257.0	14	0.90	267.5	34	0.18	254.5	21	-	-	-	-	-	-

Each project has data on defects per page found in inspections, the complexity of each module, and number of defects found in unit test (here called base test) for each block.

Hypothesis 2, uses the data presented above, and checks whether there exist a correlation between defects found during inspection/test and complexity for a module. The regression equation used to state this hypothesis can be written as:

$Y = \alpha X + \beta$, where Y is defect density, X is the complexity, and α and β are constants.

$\mathbf{H_0}$ can only be accepted if α and β are significantly different from zero and the significance level for each of the coefficients is better than 0.10. The following values were estimated:

$Y = 0.1023*X + 13.595$.

Table 10. Estimated values, Study 2

Predictor	*Estimate*	*Standard error*	*t*	*p*
β	13.595002	18.52051	0.73	0.4729
α	0.1022985	0.093689	1.09	0.2901

It indicates that the linear regression line must be rejected if a significance of level 0.10 is assumed, i.e., *neither* $\mathbf{H_2}$ *nor* $\mathbf{H_0}$ *can be refuted*. So more data is needed.

However, Ericsson reports that the best people often are allocated to develop difficult modules and more attention is generally devoted to complex software. This may explain why no significant correlation was found. More studies are anyhow needed here.

5.7 H3: Correlation between defect rates across phases and deliveries for individual documents/modules

This hypothesis, from Study 2, uses the same data as for hypothesis 2. To check for correlation between defect densities across phases and deliveries, we have analyzed the correlation between defect densities for modules over two projects. Because the lack of data in this analysis, only Project A and Project B where used (see table 9). Table 11 shows the correlation results.

Table 11. Correlation between defect density in Project A and B, Study 2.

Correlation: 0.472	**Defect density in Project A vs. Defect density in Project B**

With a correlation coefficient of 0.4672, we cannot conclude that there exists a significant correlation between the two data sets. We had only 6 modules with complete data for both projects for this test. The test should be done again, when a larger data set are available. So *neither* $\mathbf{H_3}$ *nor* $\mathbf{H_0}$ *can be refuted.*

6. Conclusion

After analysis of the data, the following can be concluded for Ericsson in Oslo:

- Software inspections are indeed cost-effective: They find around 70% of the recorded defects, take 6% to 9% of the development effort, and yield an estimated saving of 21% to 34%. I.e., finding and correcting defects before testing pays off – so "quality is free".

- 7% of the defects from inspections (3% in Study 1, 8% in Study 2) are found during the final meeting, while 93% are found during the individual reading. Almost the same distribution of defects (Major, Super Major) are found in both cases. However, Gilb's insistence on finding many (serious) defects in the final inspection meeting is not supported here.

- By comparison, [Votta93] reports that 8% of the defects are found in the final inspection meeting. Votta therefore proposes to eliminate them, since they are costly (7-14 times less cost-efficient than individual reading in our studies) and since their logistics is bothersome (binding up many busy people and thus victims to sudden cancellations). However, inspection meetings are indeed cost-efficient compared to function tests (6 times more cost-effective in Study 1), and presumably to later tests too. Inspection meetings also fulfill important social functions, like dissemination of knowledge and promotion of team spirit. At Ericsson they also serve to give an overall quality check or approval of design documents.

- Individual reading and individual desk reviews are the most cost-effective techniques to detect defects, while system tests are the least cost-effective.

- The recommended inspection rates are not really followed, since only 54% to 79% of the recommended effort is being used.

- The identified defects in a module do not depend on the module's complexity (number of states) or its modification rate, neither during inspections nor during testing.

- However, the number of defects for one concrete system (Study 1) in field-use correlated positively with its complexity and modification rate.

- We had insufficient data to clarify whether defect-prone modules from inspections continued to have higher defect densities over later test phases and over later deliveries.

- The collected, defect data has only been partly analyzed by Ericsson itself, so there is a huge potential for further analysis.

- The defect classification (Major and Super Major) is too coarse for causal analysis in order to reduce or prevent future defects, i.e. a process change, as recommended by Gilb. We also lack more precise data from Function test, System test and Field-use.

It is somewhat unclear what these findings will mean for process improvement at Ericsson. At least they show that their inspections are cost-effective, although they could be tuned wrt. recommended reading rate (number of inspected pages per person-hour, as part of overall inspection rates).

On the other hand, a more fine-grained data seem necessary for further analysis, e.g. for root-Cause-Analysis (also recommended by Gilb). More detailed information is needed on "false positives" and on overlap in detected defects among inspectors to allow capture-recapture analysis. Such defect classsification seems very cheap to implement at defect recording time, but is almost impossible to add later. However, Ericsson seems rather uninterested to pursue such changes, e.g. since "approval from headquarters" is necessary to modify the current inspection process. However, due to a change in technology platform from SDL and PLEX to UML and Java, Ericsson will anyhow have to revise their inspection process towards object-oriented technologies and corresponding inspection techniques [Travassos99].

Inspired by these findings, NTNU is anyhow interested to continue its cooperation with Ericsson on defect studies in the context of the SPIQ project. Their defect database seems under-used, so these studies may encourage a more active utilization of collected data. Further, NTNU has under way further longitudinal studies at Ericsson, spanning over several development phases and release cycles.

Acknowledgements: We thank Torbjørn Frotveit and other contacts at Ericsson for their time and interest in these investigations, and the Norwegian SPIQ project on Software Process Improvement for economic support. We also thank Oliver Laitenberger from Fraunhofer IESE, Kaiserslautern for insightful comments.

7. References

[Adams84]) Edward Adams:
"Optimizing Preventive Service of Software Products",
IBM Journal of Research and Development, (1):2—14, 1984.

[Basili96] Victor R. Basili, Scott Green, Oliver Laitenberger,
Filippo Lanubile, Forrest Shull, Sivert Sørumgård, and Marvin V. Zelkovitz:
"The Empirical Investigation of Perspective-Based Reading",
J. of Empirical Software Engineering, Vol. 1, No. 2, 1996, p. 133—164.

[Fagan76] Michael E. Fagan:
"Design and Code Inspection to Reduce Errors in Program Development",
IBM Systems J. Vol. 15, No. 3, 1976, p. 182—211.

[Fagan86] Michael E. Fagan:
"Advances in Software Inspections",
IEEE Trans. on Software Engineering, SE-12(7):744—751, July 1986.

[Gilb93] Tom Gilb and Dorothy Graham:
"Software Inspections",
Addison-Wesley, London, UK, 1993.

[Marjara97] Amarjit Singh Marjara:
"An Empirical Study of Inspection and Testing Data" (Study 1),
IDI. NTNU, Trondheim, Norway, 22 Dec. 1997.
108 p., EPOS TR 308 (diploma thesis).

[Skåtevik99] Børge Skåtevik:
"An Empirical Study of Historical Inspection and Testing Data at Ericsson" (Study 2) ,
IDI, NTNU, Trondheim, Norway, 8 Feb. 1999.
90 p., EPOS TR 350 (diploma thesis).

[Travassos99] Guilherme H. Travassos, Forrest Shull, Michael Fredericks, Victor R. Basili:
"Detecting Defects in Object Oriented Designs: Using Reading Techniques to Increase Software Quality",
Proc. Conf. on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA'99),
Denver, Colorado, 3—5 Nov.1999.
In ACM SIGPLAN Notices, Vol. 34, No. 10, Oct. 1999, p. 47—56.

[Votta93] Lawrence G. Votta:
"Does Every Inspection Need a Meeting?",
In Proc. ACM SIGSOFT 93 Symposium on Foundation of Software Engineering.
ACM SIGSOFT Engineering Notes, Vol. 18, No. 5, December 1993, p. 107—114.

Updated: 08 Dec. 1999 (rc), File: ~spiq/presentasjoner/artikler/NASA99-ericsson-v3.doc

Empirical Studies of Inspection and Test Data at Ericsson

Amarjit Singh Marjara, Cap Gemini Norway AS
Reidar Conradi, NTNU;
Børge Skåtevik, STC

NASA SEL Workshop, 1-2 Dec 1999
(rev. 8 Dec. 99)
GSFC, Washington

Agenda

- Background
- The inspection method
- Data
- Observations/questions
- Results
- Conclusions
- Recommendations

Purpose of the studies

- **$H_{1:}$** To investigate if there is a correlation between defects found during inspection/test and the complexity.
- **$H_{2:}$** To investigate if there is a correlation between the number of defects found in field-use and the complexity and the modification rate of a module.
- **$H_{3:}$** To investigate if there is a correlation between defects rates across phases and deliveries for individual documents/modules.
- A diploma work at NTNU Sept.-Dec. 1997, and diploma work at NTNU Oct'98-Feb'99, against Ericsson AS, Norway.

Background

- Data are collected at Ericsson AS, AXE-division, Oslo.
- Every development document (design, code,..) is inspected.
- Using Gilb method; an extension of Fagan's.
- Data for many projects are analysed.
- The analysed data orginates from design, unit test, function test and system test. Code is not inspected in this manner.

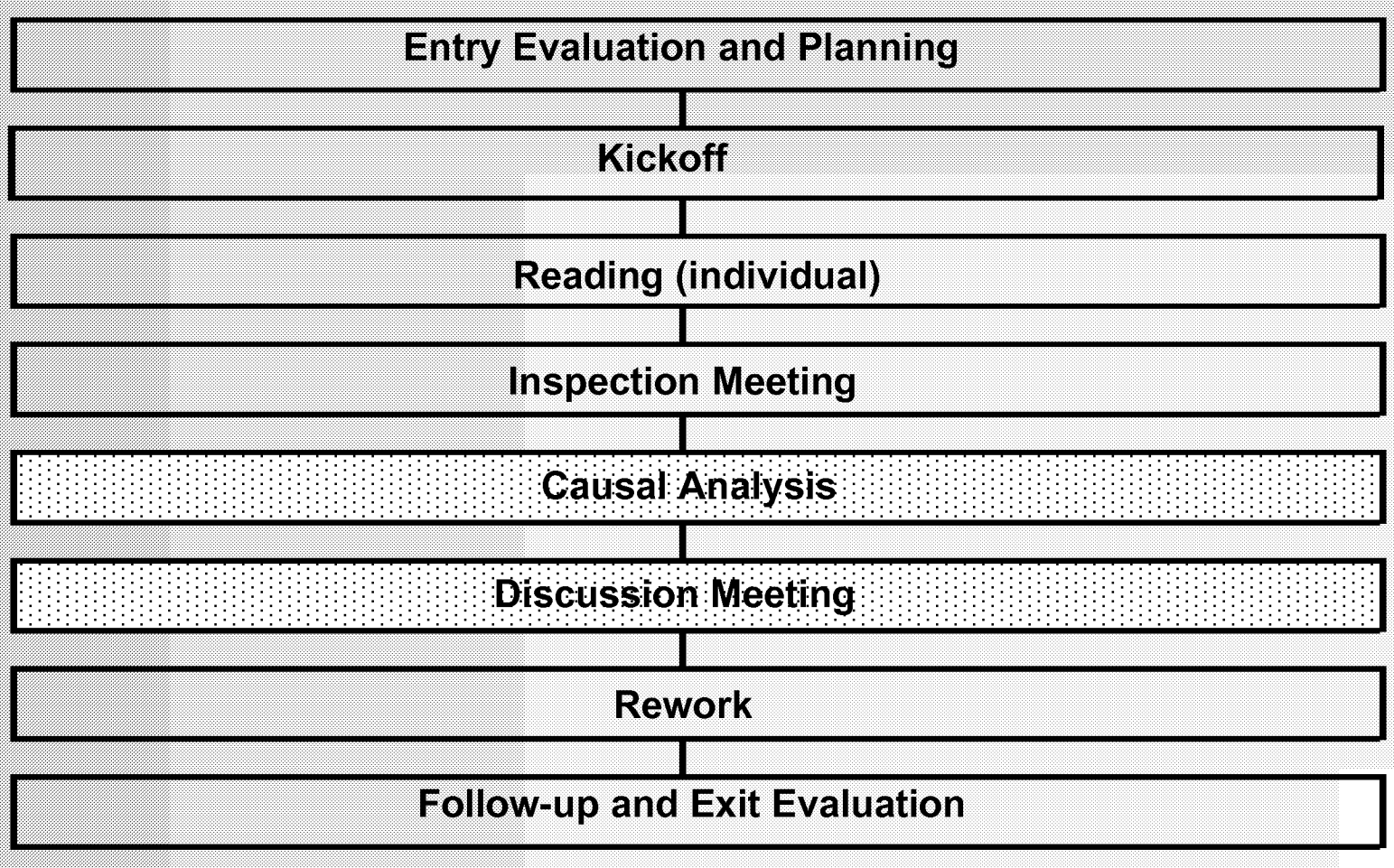
Background 2(3)

- The paper is divided in two studies:
  - Study 1:
    - Data from one project of 20.000 person-hours.
    - It includes design inspections, desk check (code review & unit test), function test, and partly system test and field-use.
    - The initial phases, such as prestudy and system study, are excluded from 20.000 ph.

Background 3(3)

- Study 2, done later:
  - A study of 6 projects ==> 100.000 ph. (including the one in Study 1).
  - Includes design inspections, code review, and unit test -- not function test etc.
  - The initial phases, such as prestudy and system study, excluded from 100.000 ph.

The inspection method 1(2)



The inspection method - 2(2)

- Provide special training for the moderators.
- Inspection meeting max two hours.
- Follow the recommended, “optimal” reading rates for the actual document type (18 such).
- Do not cover too much complex material in a single review.
- Invite the most competent inspectors to the meeting.
- Avoid personal criticisms.
- Postpone long discussions till end of the meeting.

Inspection Data

- **Block** - name of the block (module, unit).
- **Document type** - the type of document which is inspected.
- **Effort saved/lost** - estimating whether effort has been saved or not.
- **Number of persons** - doing reading and participating in the inspection meeting.
- **Planning** - effort spent on planning the inspection.
- **Kickoff meeting** - effort spent on introducing the document to the participants.
- **Reading** - total effort spent on individual preparation.
- **Inspection meeting** - group effort.
- **Rework, follow up -** defect fixing effort.
- **Defects found during reading :** classified **(Super Major or Major)**
- **Defects found in inspection meeting, pages read, pages per meeting**
  - **Defect classification: also just Super Major or Major**
- ===> **All summarized in an inspection survey,** stored in a DB

Collecting Data - testing

- Unit test
  - No. of defects found and effort spent
  - Data available per module (=unit)
- Function test, system test, field-use
  - Cause
  - Priority - seriousness of the defect
  - Data available per module
  - The effort spent on system test and field-uses not available (final integration in Stockholm).

Results

Study 1 - effectiveness of inspections and testing

- Results from Study 1:

Activity	Defects [#]	[%]
Individual reading (design)	928	61.8
Inspection meeting (design)	29	1.9
Desk check (code review + unit test)	404	26.9
Function test	89	5.9
System test	17	1.1
Field use	35	2.3
Total	**1502**	**100.0**

Note: 9.3% of defects in three latter phases.

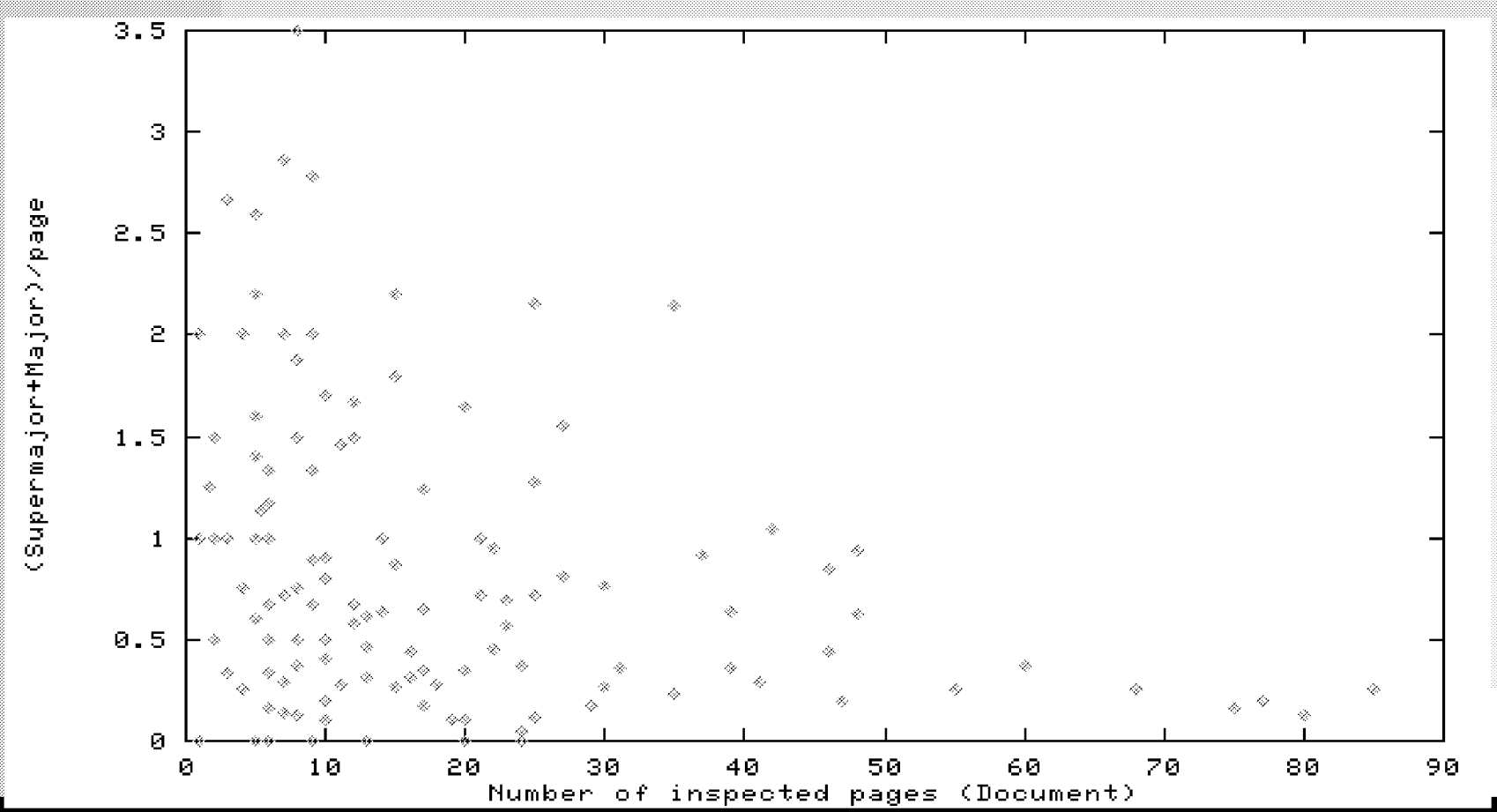
Results

Study 1- Cost-effectiveness of inspections and testing

Activity	Defects [#]	Effort [h]	Of total [%]	Effort spent to find one defect [h:m]	Estimated saved effort by early defect removal [h]
Inspection reading (design)	928	786.8	61.78	00:51	8200
Inspection meeting (design)	29	375.7	1.92	12:57	
Desk check	404	1257.0	26.91	03:07	-
Function test	89	7000.0	5.92	78:39	-
System test	17	-	1.13	-	-
Field use	35	-	2.33	-	-

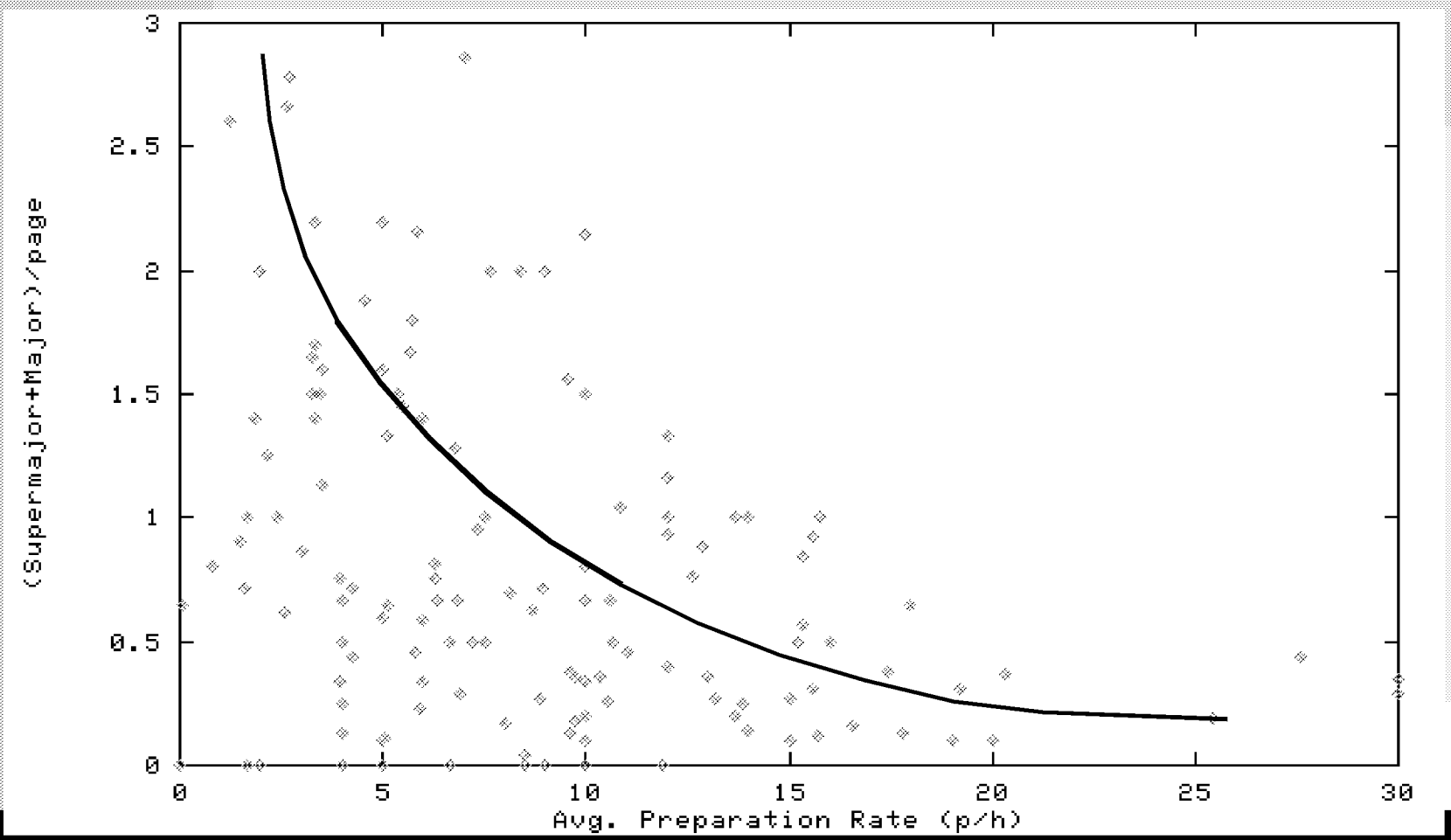
Results

Study 1 - **Number of pages inspected and defect detection rate**



Results

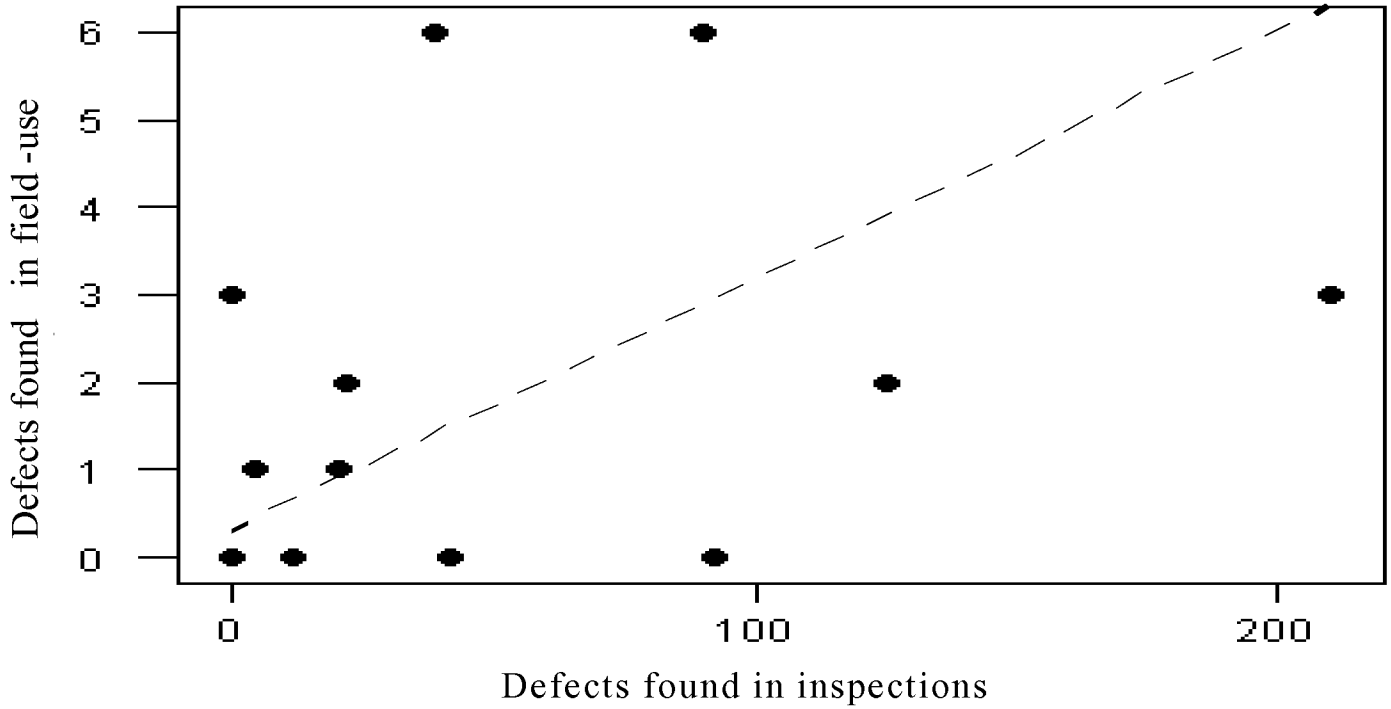
Study 1- **Reading rate and defect detection rate**



Results

Study 1- field-use defects and defects found during inspections, per module

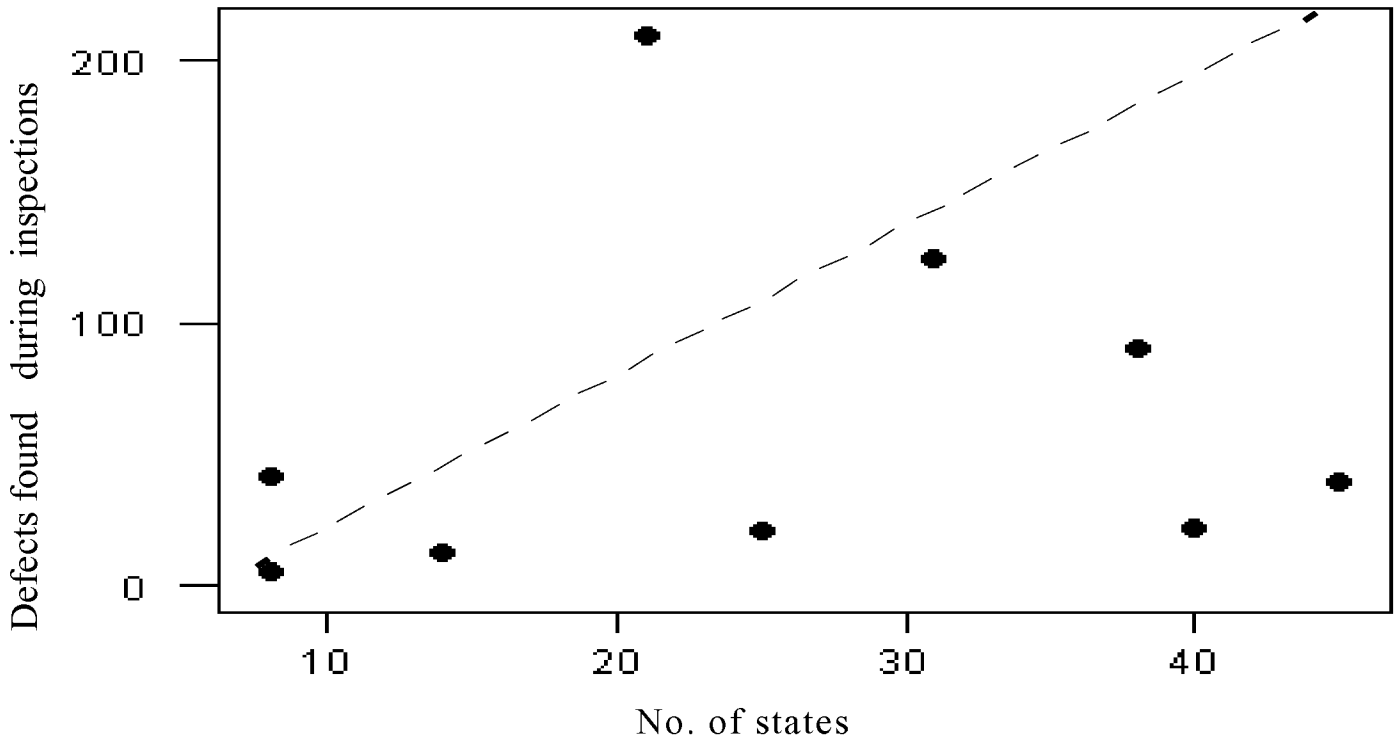
There does not seem to be a well-defined correlation between these two variables. The dashed line shows the expected (intuitively) values.



Results

Study 1: inspection defects and no. of states in a module

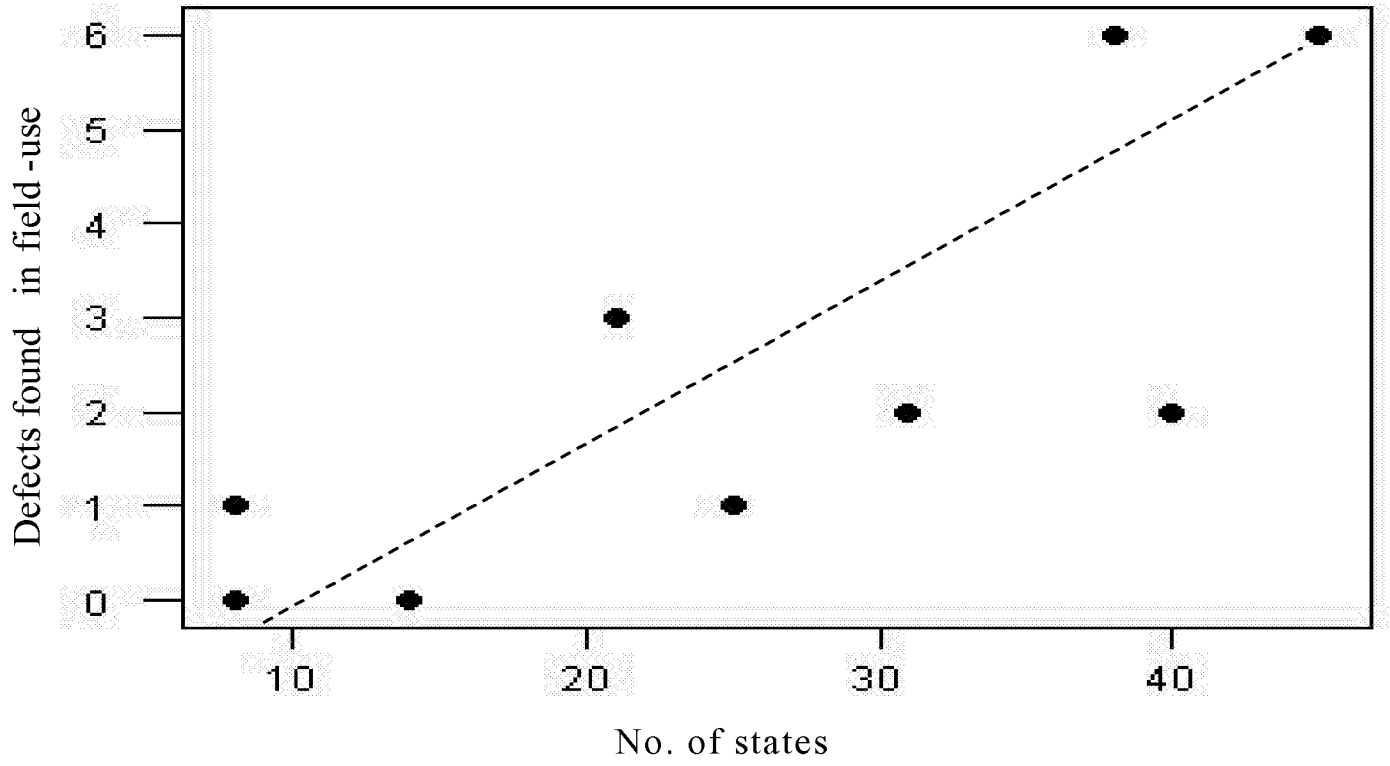
Again, there does not seem to be a well-defined correlation between these two variables. Surprisingly, the no. of defects detected in inspections seems to be rather constant if the topmost value along the y-axis is removed. Intuitively, it should be more difficult to inspect a document with a large number of states, than with a small nbumber of states. The dashed line shows the expected (intuitively) values



Results

Study 1- number of defects in field-use versus states in a module

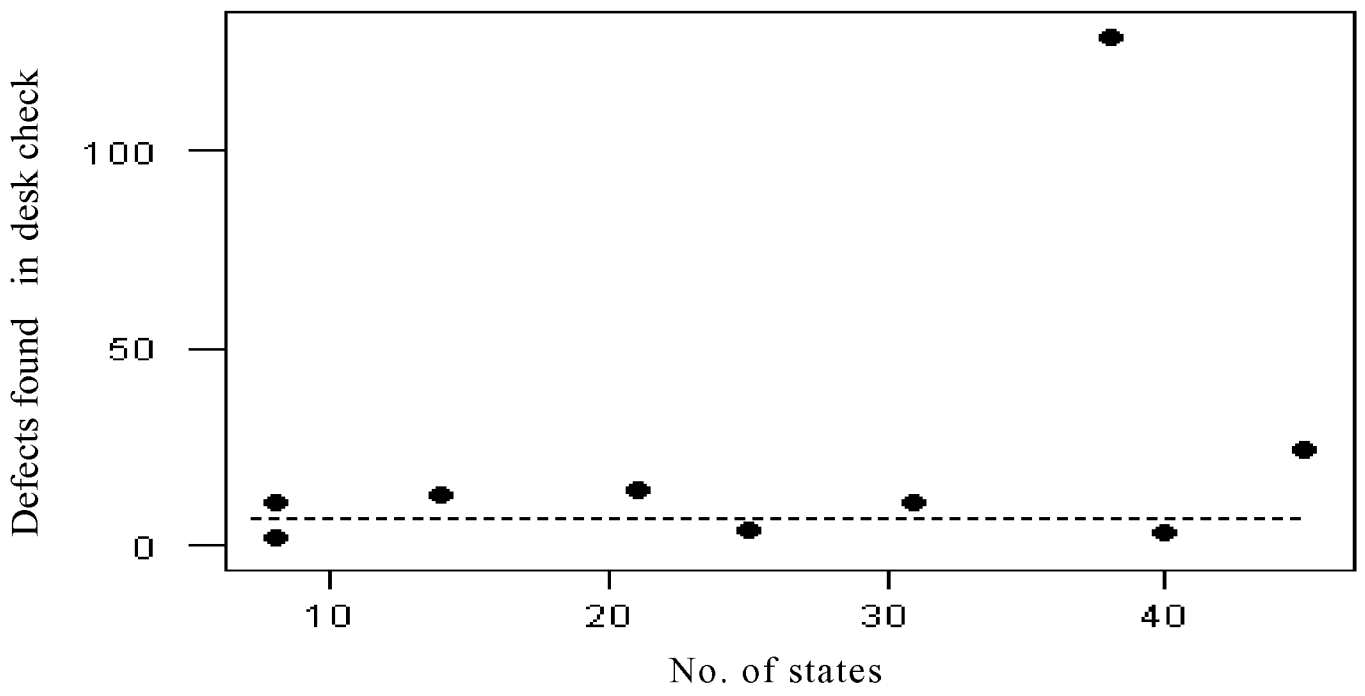
The number of states do seem to be correlated with number of defects in field-use, indicated with the dashed l
The number of system failures increases with increasing number of states. Thus, number of states represent t
inherent complexity of a module.



Results

Study 1- defects found in desk check versus states in a module

Surprisingly, the number of defects found in desk check seems to be independent of the number of states in a module. This is indicated in the figure below, if th e topmost value along the y-axis is not considered.



Results

Study 1- effort spent on inspections

- Spent: 1474 ph; whereof 1162.5 on individual reading and inspection meetings, and 311.2 of defect fixing.
- Recommended: 2723 ph, according to internal Gilb guidelines.
- Saved: 8200 ph
  - if the defects had not been detected by inspection, but detected and fixed in the later phases (not field-use).
- Inspections detect almost 65% of the registered defects, and desk check 27%. Remaining 7% in later testing activities, and 2% in field-use!!
- At Ericsson, the Gilb inspection process focuses on finding new defects in inspections, but only 3% of the defects found during inspection are actually found in the inspection meeting (”true negatives”).
  No data stored to verify ”false positives”.

Results

Study 2- effectiveness of insp. and testing

Activity	Defects [#]	[%]
Inspection reading, design	4478	71.1
Inspection meeting, design	392	6.2
Code review	832	13.2
Unit test	598	9.5
Function test etc.	?	?
Total	6300	100.0

Results

Study 2- cost-effectiveness of insp. & testing

Activity	Defects [#]	Effort [h]	Effort spent to find one defect [h:m]	Estimated saved effort by early defect removal
Inspection reading, design	4478	5563	01:15	41000
Inspection meeting design	392	3215	08:12	
Code review	832	2440	02:56	-
Unit test	598	4388	07:20	-
Function test etc.	?	?	?	?
Total	6300	15606		

Results

Study 1 and 2 - Recommended rate vs. actual total effort during inspections, including defect fixing.

Study 1: 1474 ph (54%) actually used,
out of 2723 ph recommended.

Study 2: 20,515 ph (79%) actually used,
out of 26,405 ph recommended.
0.43 defects per page.

Individual reading rate: part of total inspection rate.

All above for design documents, not source code.

Results

Study 2- how cost-efficient are inspection meetings?
Study 2- what kind of defects are found during reading and meetings?

	Individual reading	Inspection Meeting	Defect fixing	Sum effort
Effort [h]	5563	3215	11737	20515
[%]	27.12%	15.67%	57.21%	100.00%

	Major defects		Super Major defects		Sum defects	Effort	Cost-efficiency
	[#]	[%]	[#]	[%]	[#]	[h]	[h:m/defect]
In reading	4356	97.2%	122	2.7%	4478	5563	01:15
In meetings	380	96.9%	12	3.1%	392	3215	08:12
In defect log (total)	4736	97.2%	134	2.7%	4870	8778	01:48

Regression Analysis

Study 1- Hypothesis 2

The number of states ($N_{s)}$ is an important variable, because it correlates to the number of system failures (field-use defects, N_{fu}). The modification rate, $N_{mr,}$ is included in the following regression equation.

$$N_{fu} = \alpha+\beta N_s+\lambda N_{mr}$$

Here α, β, λ are contants.

Regression Analysis - Study 1

- $\mathbf{H_2}$: the fault density of a module in field-use depends on the complexity of the module (no. of states) and the its modification rate. High complexity and high modification rate will thus result in high fault density in operation.
- $\mathbf{H_0}$: the fault density of a module in field-use does not depend on the module's complexity and the modification rate.
- $\mathbf{H_0}$ is the null hypothesis, and $\mathbf{H_2}$ is the alternative hypothesis.

$\mathbf{H_2}$ can only be accepted, if β and λ are significantly different from zero and the significance level for each of the coefficients is better than 0.10.

Regression Analysis - Study 1

The following values are estimated:

$$N_{fu} = -1.73 + 0.084N_s + 0.097N_{mr}$$

Predictor	Coefficient	StDev	t	p
Constant (α)	-1.732	1.067	-1.62	0.166
States (β)	0.084	0.035	2.38	0.063
Modrate (λ)	0.097	0.034	2.89	0.034

The estimated values for the coefficients are given above, along with the std. deviations, t-value for testing if the coefficient is 0, and the p-value for this test.
$S = 1.200$; $R^2 = 79.9\%$, $R^2_{(adj)} = 71{,}9\%$.

The analysis of variance is summarised below:

Source	DF	SS	MS	F	P
Regression	2	28.68	14.34	9.96	0.018
Error	5	7.20	1.44		
Total	7	35.88			

It should be noted that the constant is not significant, but the states and modrate are significant. H_2 can be accepted.

Regression Analysis

Study 2- Hypothesis 1

Hypothesis 1 uses the data presented above, and checks whether there exist a correlation between defects found during inspection/test and complexity for a module.

The regression equation used to state this hypothesis can be written as:

$Y = \alpha X + \beta$, where Y is defect density, X is the complexity and α, and β are constants.

$\mathbf{H}_1$ can only be accepted if α and β are significantly different from zero and the significance level for each of the coefficients is better than 0.10.

Regression Analysis
Study 2- Hypothesis 1

- The following values was estimated: Y = 0.1023*X + 13.595

Predictor	*Estimate*	*Standard error*	*t*	*p*
β	13.595002	18.52051	0.73	0.4729
α	0.1022985	0.093689	1.09	0.2901

It indicates that the linear regression line must be rejected if a significance of level 0.10 is assumed, i.e. *neither* $\boldsymbol{H_0}$ *nor* $\boldsymbol{H_1}$ *can be refuted*, reason being too few data.

Regression Analysis
Study 2- Hypothesis 3

- To check for correlation between defect densities across phases and deliveries, we have analyzed the correlation between defect densities for modules over two projects.

Correlation: 0.4672	**Variable 1: Defect density – Project A**	**Variable 2: Defect density - Project B**

Thus, no significant correlation found between the two data set.
But only 6 modules with complete data for both projects for this test.
So neither $\mathbf{H_0}$ *nor* $\mathbf{H_3}$ *can be refuted*.

Conclusions 1(2)

- Inspections are the most cost-effective in defect detection; function testing etc. the least cost-effective.
- Inspections find ca. 70% of recorded defects, takes 6--9% of development effort. **Estimated net saving: 34--21%!!**
- 7% of inspection defects (Study 1: 3% , Study 2: 8%) found during final meeting, 93% during individual reading. But insp. meetings more cost-effective than function test!
- Individual reading & code reviews: the most cost-effective techniques to detect defects.
- Recommended inspection rates not followed, only 54--79% of recommended effort spent.

Conclusions 2 (2)

- Defect density of a module in field-use depends on its complexity and modification rate.
  Generally, low-complexity designs have lower defect rates.
  But: **is it possible to create low complexity designs for real-time telecom software?**
  Pay extra attention when designing complex parts?
- Ericsson has focussed on finding new defects during meetings; only 7% of inspection-phase defects found here.
- The defect classification is too coarse.
- Must record ”false positives”; now only ”true negatives”.
- Must record overlap between inspectors, to facilitate capture-recapture analysis

Recommendations

Record and analyze more data properly, e.g. to check if:

- Defect-prone modules during inspections also are defect-prone during later tests and field-use (**longitudal** analysis)?

- One defect type dominates in one project, or if subsequent projects will have the same defect types? Will need **better defect classification** (ISO/IEEE)!

- We may **omit inspection meetings** for some document types or try **virtual inspection meetings** on the net/web.

Session 8: COTS

Steven Demurjian, University of Connecticut

Daniil Yakimovich, University of Maryland

JINI: A Technology for 21st Century -- Is it Ready For Prime Time?*

Prof. Steven A. Demurjian, Sr.
Computer Science & Engineering Dept.
The University of Connecticut
Storrs, CT 06269-3155
steve@engr.uconn.edu
Tel: 860.486.4818
Fax: 860.486.4817

Dr. Paul Barr
The MITRE Corp
145 Wyckoff Road
Eatontown, New Jersey 07724
poobarr@mitre.org
Tel: 732-935-5584
Fax: 732-544-8317

1. Introduction and Motivation

Distributed computing applications for the 21st century are network centric, operating in a dynamic environment where clients, servers, and the network itself all have the potential to change drastically over time. A *distributed application*, a system of systems, must be constructed, consisting of legacy, commercial-off-the-shelf (COTS), database, and new client/server applications that must interact to communicate and exchange information between users, and allow users to accomplish their tasks in a productive manner. The issue is to promote the use of existing applications in new and innovative ways in a distributed environment that adds value. To adequately support this process, the network and its software infrastructure must be an active participant in the interoperation of distributed applications. Ideally, we are interested in distributed applications that plug-and-play, allowing us to plug in (and subtract) new "components" as needs, requirements, and even network topologies change over time.

JINI [Arno99, JINI, JINIARCH] is a new architecture built on top of Java's remote method invocation (RMI) that promotes the construction and deployment of robust and scalable distributed applications in a network centric setting. JINI technology is forcing software designers and engineers to abandon the client/server view in order to adopt a *client/**services*** view. In JINI, a distributed application is conceptualized as a set of services (of all resources) being made available for *discovery* and use by clients. To accomplish this, JINI makes use of a *lookup service*, which is essentially a registry for tracking the services that are available within a distributed environment. Services in JINI *discover* and then *join* the lookup service, registering the services (of each resource) that are to be made available on the network. Thus, JINI is conceptually very similar to a distributed operating system, in the sense that resources of JINI are very similar to OS resources. However, in JINI these resources can be dynamically defined and changed. To illustrate JINI, consider that a service `register_for_course(course#)` for a Course database in a University application may be registered with the lookup service. Clients request services by interacting with the lookup service, e.g., asking for `register_for_course(CSE230)`. The lookup service returns a proxy to the client for the location of the service. The client then interacts directly with the service via the proxy to execute the service, e.g., registering for CSE230. In this process, there are a number of important observations. First, services can come (register and join) and go (leave) without impunity, since all interaction with services occurs via the lookup service. Second, clients locate and utilize services without knowing their location on the network, allowing clients to work without interruption as long as "some" service can be located to meet their needs. Third, the location of clients and/or services on the network can change at any time without impacting the network or the users.

Our efforts are motivated from two perspectives. First, by Army requirements, we evaluated the JINI technology in support of present and future systems. Second, as part of grant from AFOSR on large-scale, multi-agent, distributed mission planning and execution in complex dynamic environments, we have been considering the ability of software agents (written using Java) to interact with JINI resources and services. In both efforts, there are a number of common, fundamental questions:

- **Can JINI Support Highly-Available Distributed Applications?**
- **Can JINI Support an Environment with Dynamic Clients and Replicated Services?**
- **Will Clients Continue to Operate Effectively if Replicated Services Fail?**
- **Can JINI be Utilized to Maintain "minutes-off" Data Consistency of Replicas?**
- **Is JINI Easy to Learn and Use? What is Maturity Level of JINI Technology?**

* The work in this paper has been partially supported by a contract from the Mitre Corporation (Eatontown, NJ) and AFOSR research grant F49620-99-1-0244.

The reality is that new technologies offer new challenges, with the potential to reap benefits if adopted. However, for future Army systems, it is important that a careful balance is drawn to opt for mature technologies while targeting emerging technologies with potential. The key issue is where JINI fits – as a mature technology or yet another one with potential? The remainder of this abstract reviews JINI, our experimental prototyping effort, summarizes our results, and proposes a series of future work to answer the question: **"Is JINI Ready for Prime Time?"**

2. JINI

Stakeholders (software architects, designers, and implementors) can utilize JINI to construct a distributed application by federating groups of users (clients) and the resources that they require. In JINI, the resources register *services* which represent the functions that are provided for use by clients (and other services). In a sense, the services are similar in concept to public methods that are exported for usage as part of an applications class library (API). JINI is versatile, and allows a service to represent any entity that can be used by a person, program (client), or another service, including: a computation, a persistent store, a communication channel, a software filter, a real-time data source (e.g., sensor or probe), a hardware device (e.g., printer, display, etc.), and so on. The services are registered with a *look-up service.* The registration of services occurs using a *leasing* mechanism. With leasing, the services of a resource can be registered with the lookup service for a fixed time period or forever (no expiration). The lease must be renewed by the resource prior to its expiration, or the service will become unavailable. This feature, in part, supports high availability, since it requires the resources to constantly reregister their services; if a resource goes down and does not reregister, the leases on its services expire, and the services will then be unavailable from the lookup service.

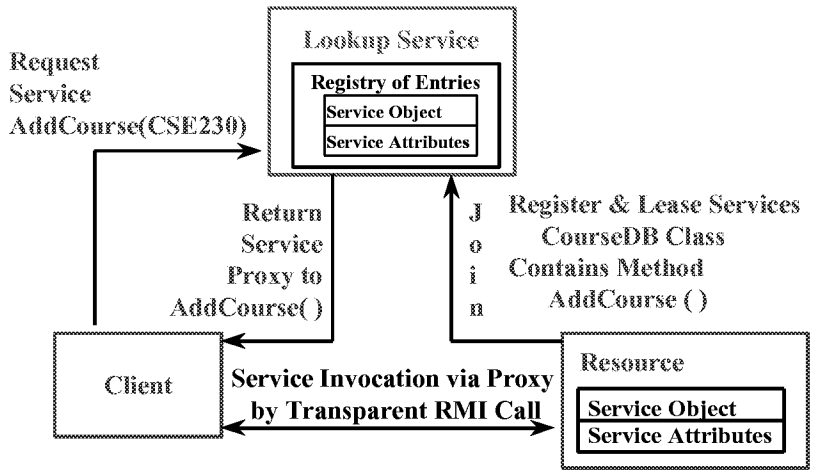
As a technology, JINI provides an infrastructure to design and construct distributed applications with a network centric approach that assumes an environment where there is a requirement for the spontaneous interaction of clients and services. Spontaneity from a client perspective supports the dynamic behavior of clients, where they enter and leave the network unpredictably. While connected, clients are guaranteed that either the visible services are available or that failure can be trapped and handled. Spontaneity from a resources perspective, means that when resources fail, the network can adapt, to insure that redundant services, if available, are now accessible to clients. Operationally, when a client wishes to interact with a service, the interaction can occur by either a download of code from service to client, or the passing of a proxy which allows a RMI-like call by the client to the service.

The lookup service is the clearinghouse of a JINI network centric application, since all interactions by resources (e.g., discovering lookup services, registering services, renewing leases, etc.) and by clients (e.g., discovering lookup services, searching for services, service invocation, etc.) must occur through the lookup service. When there are multiple lookup services running on a network, it is the responsibility of the resources to register with them (if relevant). Clients can interact with multiple lookup services, and in fact, it is possible for groups of clients to be established that will always consult a particular "close" lookup service, dictated perhaps by network topology or traffic. Whenever resources leave the environment (either gracefully or due to failure), the lookup service must adjust its registry. There is a time lag between the resource leaving and the removal of services from the registry. Clients must be sophisticated enough to be able to dynamically adjust to these situations.

After discovery has occurred, the resources register services on a class-by-class basis. The class is registered as a *service object* which contains a Java programming interface to the service, namely, the public methods available to clients coupled with a set of optional descriptive service attributes. This registration process is referred to *joining* and is shown in Figure 1. In JINI terms, the service object is registered as a *proxy*, which contains all of the information that is needed to invoke the service. In the request for service, shown in Figure 1, a client will ask for the service to register for a course of the CourseDB class based on the signature of the method: `status register_for_course(int)`. The lookup service will return a service proxy that allows the client to invoke any or all of the methods defined within the service. Using the proxy, the client invokes the needed method(s) as it would any other Java method; the call transparently utilizes RMI with the result of the call returned to the client. The interaction between the client and the resource occur independent from the lookup service.

A lease is the part of the JINI programming model that allows the resources to set the limits of its utilization of services, and allows the lookup service to remove services from its registry that are no longer available. A resource can lease a service to a lookup service forever (not recommended) or lease with a specific expiration date (in milliseconds). If leased using an expiration date, the resource is responsible for renewing the lease prior to its expiration. The leasing and renewal process is intended to keep the registry fresh, containing all active and working

services. This is of particular importance in a distributed application where resources leave the network due to failure or other reasons. When a resource leases its services with specific expiration times, if failure occurs, when the lease expires and is not renewed, the services will no longer be available. In addition, the lookup service periodically checks to see if services (and resources) are active. Whenever failure occurs, there is a time period when services will be listed in the registry that are unavailable to clients, and in fact clients will receive exceptions if they try to execute such services. Thus, even if a client receives the proxy for a service that is active in the registry, there is no guarantee that the service will be available when invoked. Thus, it is imperative that software engineers design clients that are able to handle this situation.



1. Client Invokes AddCourse(CSE230) on Resource
2. Resource Returns Status of Invocation

Figure 1: Join, Lookup, and Invocation of Service.

3. Experimental Prototyping Effort

We have taken an experimental prototype approach to evaluate the capabilities of JINI under WinNT to determine if JINI is "ready for prime time". The goal of the experimentation is to explore the ability of JINI to support applications that require high availability (via replication of resources and their services and data) in an environment where the replicated resources are volatile. Clients, which are also entering and leaving the network, consult the JINI lookup service to locate and subsequently execute the "services" of the replicated resource that are necessary to carry out their respective tasks. If one of the services fails, there is a back-up service that can be utilized to support the client. The replicated databases must be kept consistent, but at any given time point, the data in one database might be "minutes off" the data in the other databases. Over time the databases will synchronize and contain the same information. It is crucial that updates not be lost during the modification and synchronization processes.

A total of six experimental prototypes have been developed modeled on a university application where Persons (students and faculty) are attempting to access and/or modify information related to a course schedule. Students and faculty have a GUI (Java client application) through which they must enter their name and password, and once verified, are able to access course information. To support this, both a PersonDB (for authentication and authorization) and a CourseDB must be available. These two databases are stored in Microsoft Access, and a Java application or database resource, offers a set of "services" that are made available by registration with JINI to clients. A Java GUI client consults the JINI lookup service to search for appropriate services of the replicated database resource that can satisfy their requirements as needed by the student/faculty request. Whenever a Java GUI client modifies the CourseDB as a result of a user request, all other replicated CourseDBs must be modified so that the replicas remain consistent. However, there may be a time difference where the data in one CourseDB is minutes off the data in the other CourseDBs. For discussion purposes, Prototype 6 is shown in Figures 2 and 3.

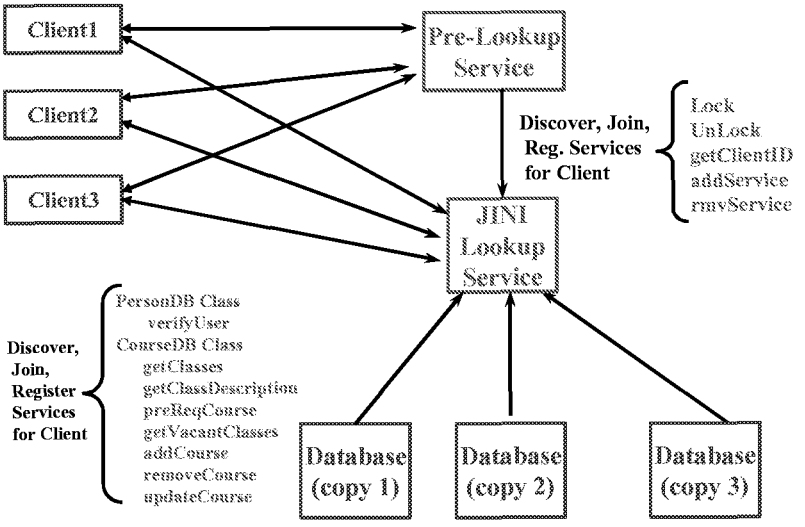


Figure 2: Pre-Lookup Services in Prototype 6.

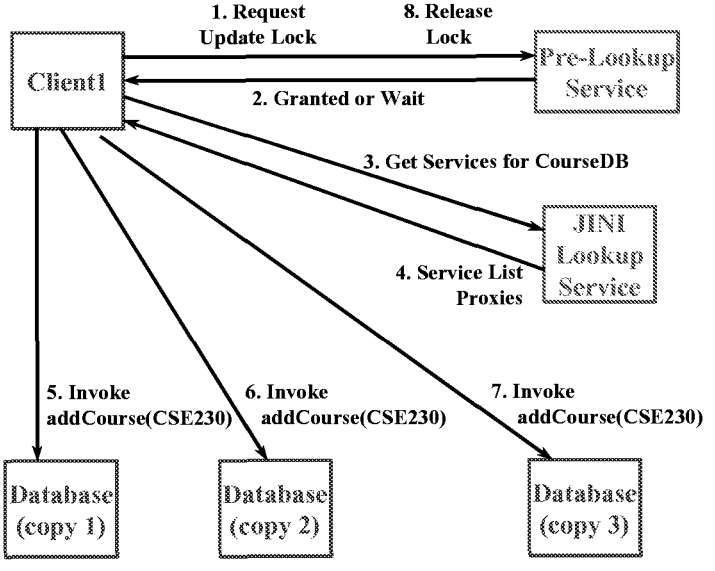


Figure 3: Execution Process in Prototype 6.

Prototype 6 incorporates a pre-lookup resource and associated services that implements a protocol that supports simultaneous reads in conjunction with at most one exclusive write, and includes PersonDB and CourseDB services for use by GUI clients. The pre-lookup services as shown in Figure 2, allow the locking and unlocking of services, identify clients (`getcClientID`), and permit replicated database resources to register their services with the pre-lookup service (`addService` and `rmvService`). Thus, clients can still read the data even if one client is holding a write lock. PersonDB services are for authorization and authentication of the client, while CourseDB services allow course information to be queried and changed. Figure 3 illustrates the process and steps taken by a client. After startup, the client applications will be interested in discovering and utilizing services. In Prototype 6, prior to the JINI lookup service being consulted, the client must first interact with the pre-lookup service, as shown in Figure 3, arrow 1. The client consults with the pre-lookup service by discovering its existence and interacting with the JINI

lookup service to obtain a proxy to request a lock. If a lock on the required service (read, insert, delete, or modify the CourseDB) is granted, the client can proceed according to arrows 3 through 7 in Figure 3. If a lock is not granted, the client is told to wait. The pre-lookup service will queue the client's identifier for the requested service to insure that starvation is prevented for clients that are denied locks at the pre-lookup service. Then, Client 1, in this case, enters a loop which will continuously request the lock (arrow 1) from the pre-lookup service. As long as another client holds the lock, a wait response will be sent to Client 1. Eventually the client holding the lock desired by client 1 will release the lock. When Client 1 next requests the lock and the first element of the queue for the service contains its identifier, Client 1 will be granted the lock, and processing proceeds via arrows 3 through 7.

4. Conclusions and Recommendations

Our conclusions and recommendations are constructed from a two-fold perspective. First, our efforts on the experimental prototypes have answered, in part, the questions posed in the introduction, specifically:

- **Can JINI Support Highly-Available Distributed Applications?** Yes, in fact Prototype 6 demonstrates that JINI can be utilized to architect solutions that are highly available.
- **Can JINI Support an Environment with Dynamic Clients and Replicated Services? Will Clients Continue to Operate Effectively if Replicated Services Fail?** Yes, in Prototype 6, it was possible to start and stop clients and stop and start resources. As long as JINI was given time to remove "failed" services, the clients and resources continued to interact effectively.
- **Can JINI be Utilized to Maintain "minutes-off" Data Consistency of Replicas?** Prototype 6 with the pre-lookup guaranteed that no updates would be lost if different clients attempted simultaneous updates.

The results are extremely relevant for present and future Army systems, and for distributed enterprise applications, in general, since the different architectural components of the prototypes can be cast as a new Java GUI, a legacy relational databases wrapped using JDBC/ODBC, and databases for authorization and general purpose information of interest to clients.

Second, is **JINI Ready for Prime Time?** That is clearly the question of interest. In our limited, yet concentrated evaluation of JINI, we have found many features that make it extremely attractive as a 21st century technology. Our reasons for believing JINI is ready for prime time include:

1. **Compatibility of JINI with Java write once run anywhere infrastructure**. The Java language and environment under which JINI operates is extremely homogenous, is operating system independent, and promotes interoperability between all of the components (clients and services) within the distributed application.
2. **Commitment of Sun to Java and JINI technologies, as evidenced by a recent keynote address by Chief Scientist Bill Joy [BJOY].** There is a significant commitment to JINI by Sun, and an expectation that JINI will play a major role in the Java arena in the coming years.
3. **Understandability and ease of use of JINI**. The individuals doing development had Java and database expertise, but no background in using JINI, Visual Café, and JDBC/ODBC. In 400 hours of work over the two month period of the work, six prototypes were designed and developed This speaks to the ease of use of Java and JINI technologies.
4. **High-level abstraction nature of JINI API.** From a software engineering perspective, one of the major strengths of JINI is the ability to design a solution to a distributed application in terms of clients and the services that are required. This design can be constructed using a UML modeling tool. We believe that with JINI, UML modeling tools, and Java development environments, good software engineering practices and products can be attained.

However, our enthusiasm must also be tempered by the fact that our investigation, exploration, and evaluation of JINI is only in the initial stages. While our experiences have been mostly positive, there are a number of future work topics that must be explored in detail to arrive at a definitive conclusion.

- **Interoperability of JINI with critical technologies.** Will JINI work with legacy, COTS, and database assets? Will JINI inter-operate with CORBA and other distributed computing solutions? Can JINI and software agent paradigms successfully interact? All are critical to assess JINI's utility in 21st century.

- **Verification of write-once-run-anywhere.** Is prototype of Section 3 extensible to Win95/98 and Solaris? Will Oracle, Informix, and other database platforms work? JINI's readiness for 21st century must be verified by conducting multi- and heterogeneous platform experiments.
- **Utility/robustness of other JINI technologies.** The list includes two-phase commit transactions, events in JINI, JINI's security model, and JavaSpaces, an API on top of JINI.
- **High-availability via multiple lookups and pre-lookup services.** Great care must be taken to explore, design, and implement prototypes that allow the incorporation of multiple lookup/pre-lookup services to have a reasonable and manageable impact on client applications.
- **Performance and scalability.** While our prototypes worked with 3 NTs, in practice, 10s, 100s, and even 1000s of clients and resources will need to interact. Consequently, the ability of JINI to scale and maintain performance in such a situation will be crucial.

Also, it is important to note that the JINI specification continues to evolve [JINISPEC]. Despite this cautionary note, based on our experiences and intuition, we believe that JINI has great promise and will be a successful and useful technology for the 21st century.

References

[Arno99] K. Arnold, et al., *The JINI Specification*, Addison-Wesley, 1999.

[Edwa99] K. Edwards, *Core JINI*, Prentice-Hall, 1999.

[Free99] E. Freeman, et al., *JavaSpaces Principles, Patterns, and Practice*, Addison-Wesley, 1999.

[Morr97] M. Morrison, et al., *Java Unleashed*, second edition, Sams.net Publishing, 1997.

[Wald99] J. Waldo, "The JINI Architecture for Network-Centric Computing", *Communications of the ACM*, Vol. 42, No. 7, July 1999.

[BJOY] http://www.javasoft.com/features/1999/07/bill.joy.html

[JINI] http://www.sun.com/jini/

[JINIARCH] http://www.sun.com/jini/whitepapers/architecture.html

[JINISPEC] http://www.sun.com/jini/specs/jini1_1spec.html

Sample JINI Software:

http://www.enete.com/download/ and http://wwww.artima.com/javaseminars/modules/Jini/CodeExamples.html

http://www.jinivision.com and http://members.home.net/jeltema

JINI Tutorial:

http://pandonia.canberra.edu.au/java/jini/tutorial/Jini.xml

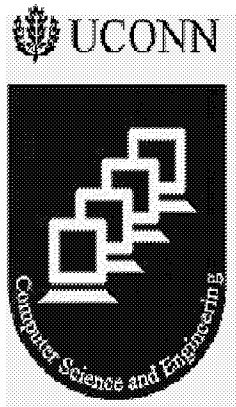
JINI-Related Information and Links:

http://www.jini.org and http://www.eli.sdsu.edu/courses/spring99/cs696/notes/index.html

http://www.artima.com/objectsjini/introJini.html and http://www.artima.com/jini/resources/index.html

Link for JINI Installation:

http://developer.java.sun.com/developer/products/jini/installation.html



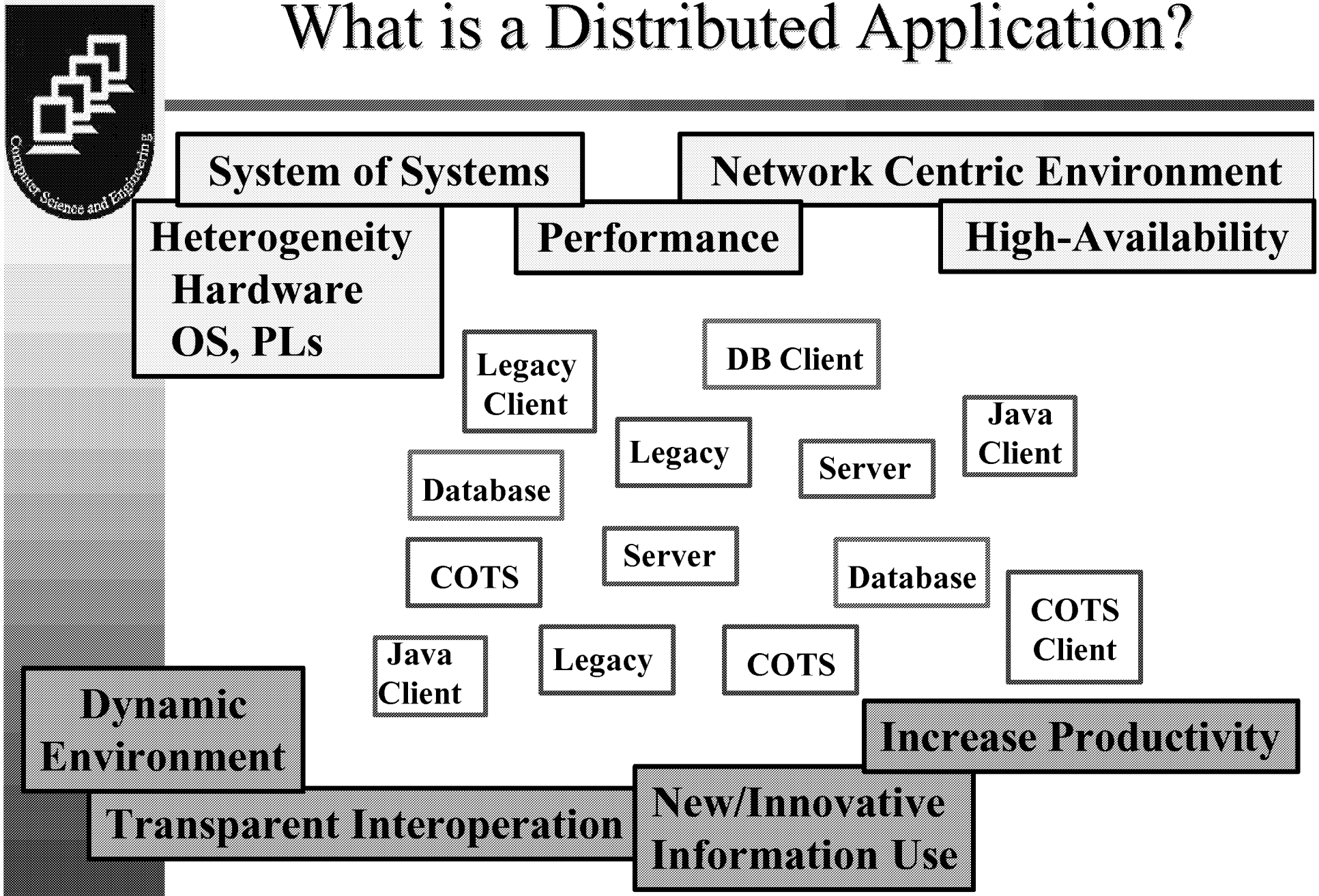
JINI: A Technology for 21st Century Is it Ready for Prime Time?

Prof. Steven A. Demurjian, Sr.
Computer Science & Engineering Dept.
191 Auditorium Road, Box U-155
The University of Connecticut
Storrs, Connecticut 06269-3155
steve@engr.uconn.edu
www.engr.uconn.edu/~steve
Tel: 860-486-4818
Fax: 860-486-4817

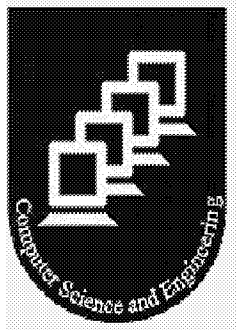
Dr. Paul Barr
The MITRE Corp
145 Wyckoff Road
Eatontown, New Jersey 07724
poobarr@mitre.org
Tel: 732-935-5584
Fax: 732-544-8317

UCONN

What is a Distributed Application?

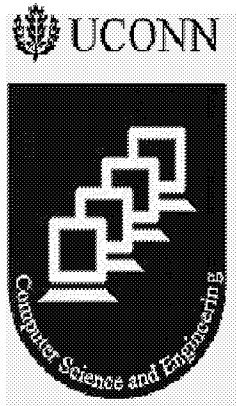


UCONN



Goals of Research Effort

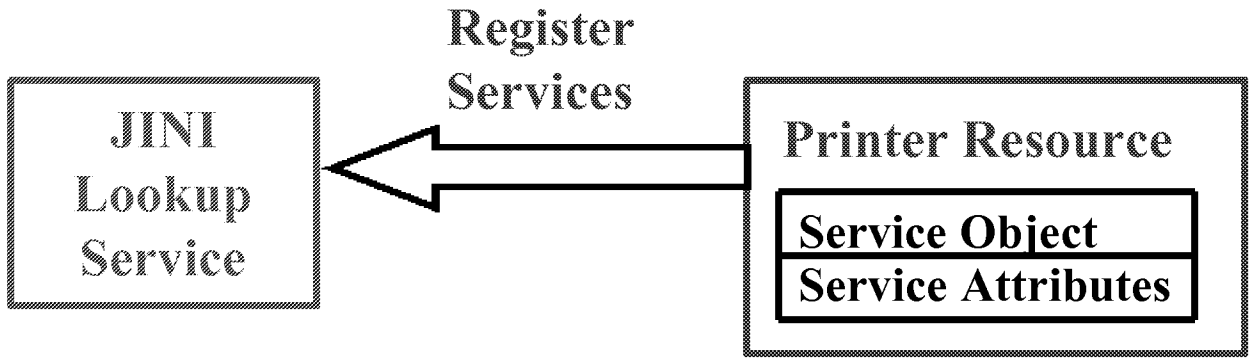
- Can JINI Support **Highly-Available Distributed Applications**?
- Can JINI Support a Network-Centric Environment with **Dynamic Clients and Services**?
- Will Clients Continue to Operate Effectively if **Replicated Databases Services Fail**?
- Can JINI be Utilized to **Maintain "minutes-off" Data Consistency** of Replicas?
- Is JINI Easy to Learn and Use?
- What is Maturity Level of JINI Technology?
- **Is JINI Ready for Prime Time???**



Sun's JINI Technology
Key JINI Concepts and Terms

- A *Resource* Provides a Set of Services for Use by Clients (Users) and Other Resources (Services)
- A *Service* is Similar to a Public Method
  - Exportable - Analogous to API
  - Any Entity Utilized by Person or Program
  - Samples Include:
    - Computation, Persistent Store, Comm. Channel
    - Software Filter, Real-Time Data Source
    - Sensor or Probe, Hardware (Printer, Display, etc.)
    - **Anything that is Relevant for Your Domain!**
- Services Register with *Lookup Service*
  - **Clearinghouse** for Resources to Register Services and Clients to Locate Services

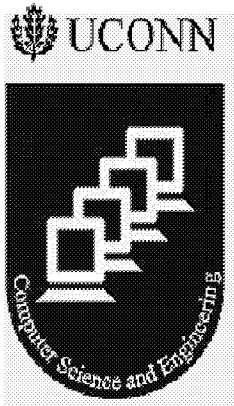
Sun's JINI Technology Resources & Services



- Sun's Initial Perspective
  - JINI for Hardware
  - Printers, Digital Cameras, etc.
  - Plug-and-Play on Network
- Services Registered
  - Class by Class Basis
  - Granularity up to SWE

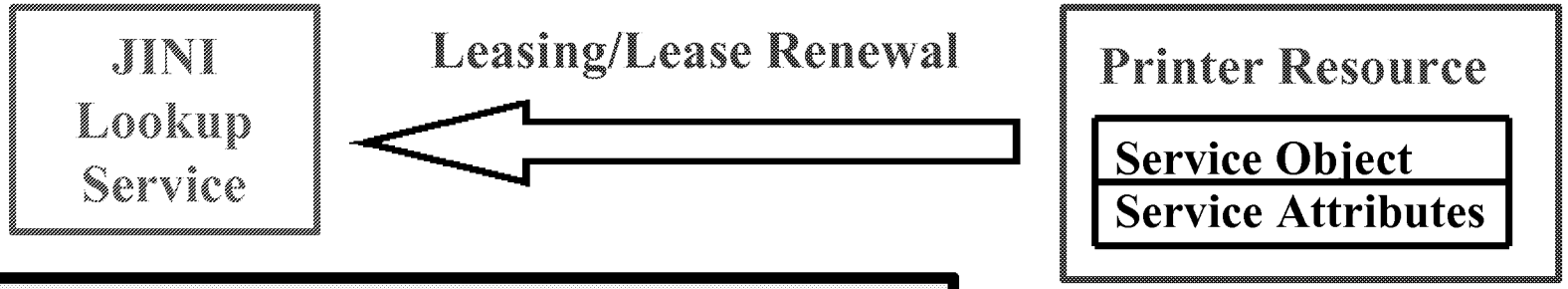
Class and Methods Define Services to be Registered

PrinterActions Class
- **enqueuePrintJob**
- **dequeuePrintJob**
- **getPrinterStatus**
- **getPrinterType**
- **installPrinter**
- **removePrinter**
- **startJob**
- **cancelJob**



Sun's JINI Technology Registration & Leasing

- FOREVER or EXPIRATION DATE (millisecs)
- Renewal Must Occur Prior to Expiration
- JINI Provides Lease Renewal Manager to Allow Resource to Delegate Renewal Responsibility



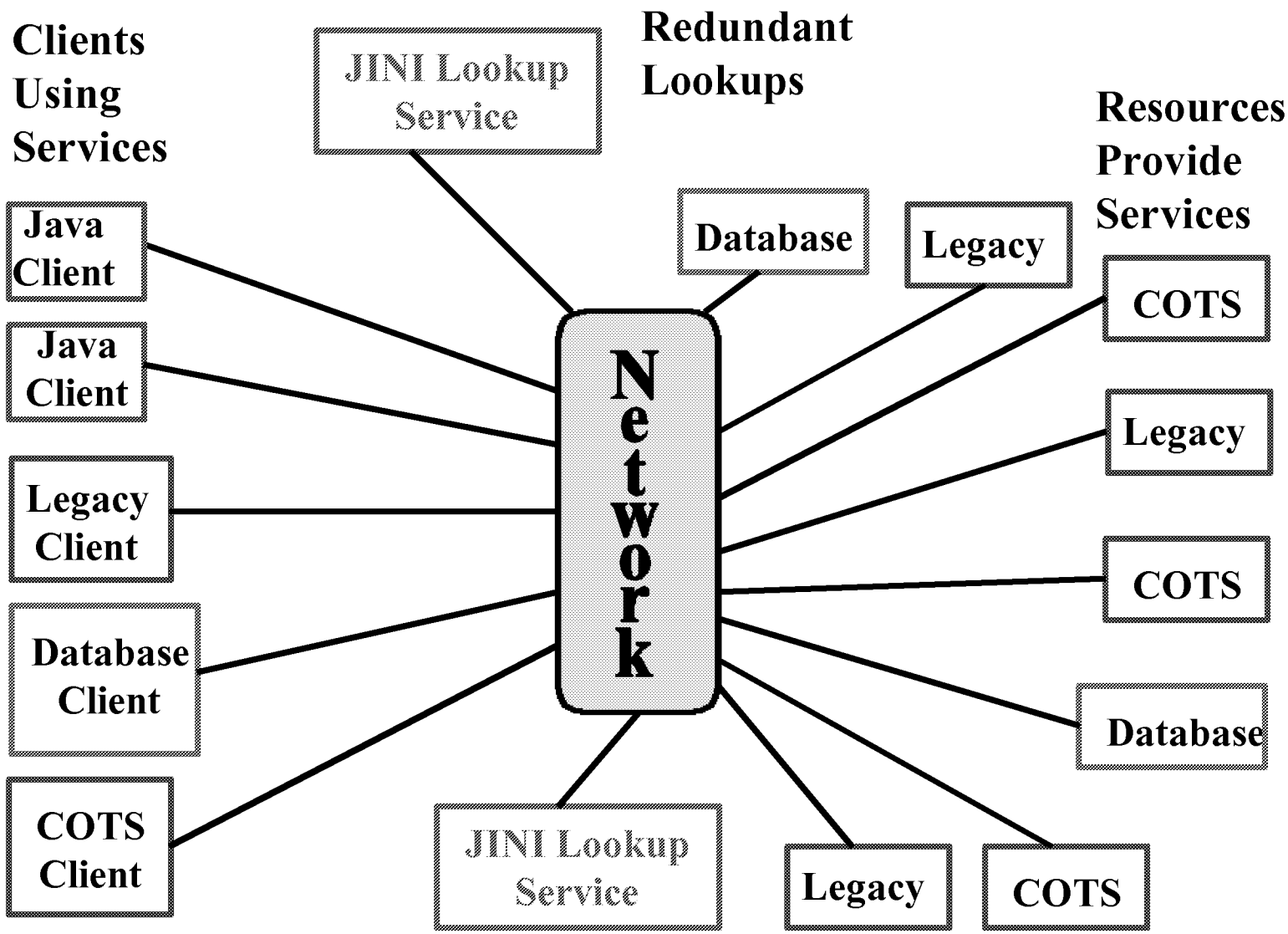
Lease for 5 minutes (300000 msec)
Must Renew Before 5 Minutes Expire
If Not Renewed, Lookup Removes
If Failure, Lookup May Still Supply Service Until Expiration (5 mins)
Client MUST be SMART!

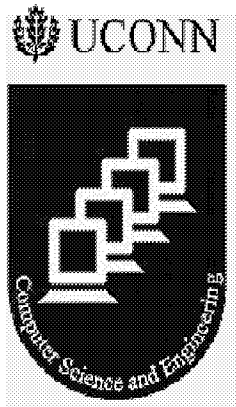
Class and Methods Define Services to be Registered

- PrinterActions Class
  - enqueuePrintJob
  - dequeuePrintJob
  - getPrinterStatus
  - getPrinterType
  - installPrinter
  - removePrinter
  - startJob
  - cancelJob

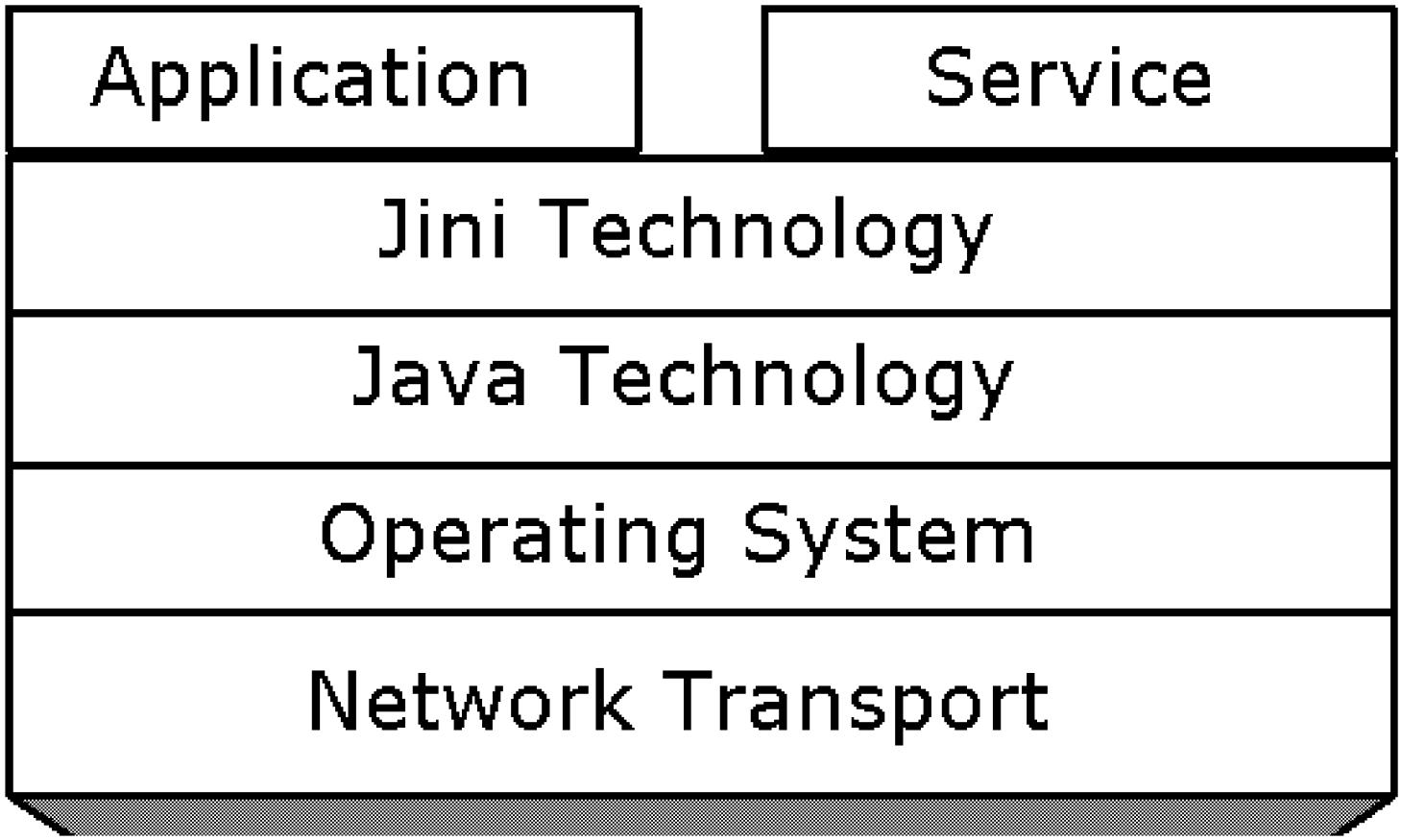
UCONN

Sun's JINI Technology Support for Distributed Computing



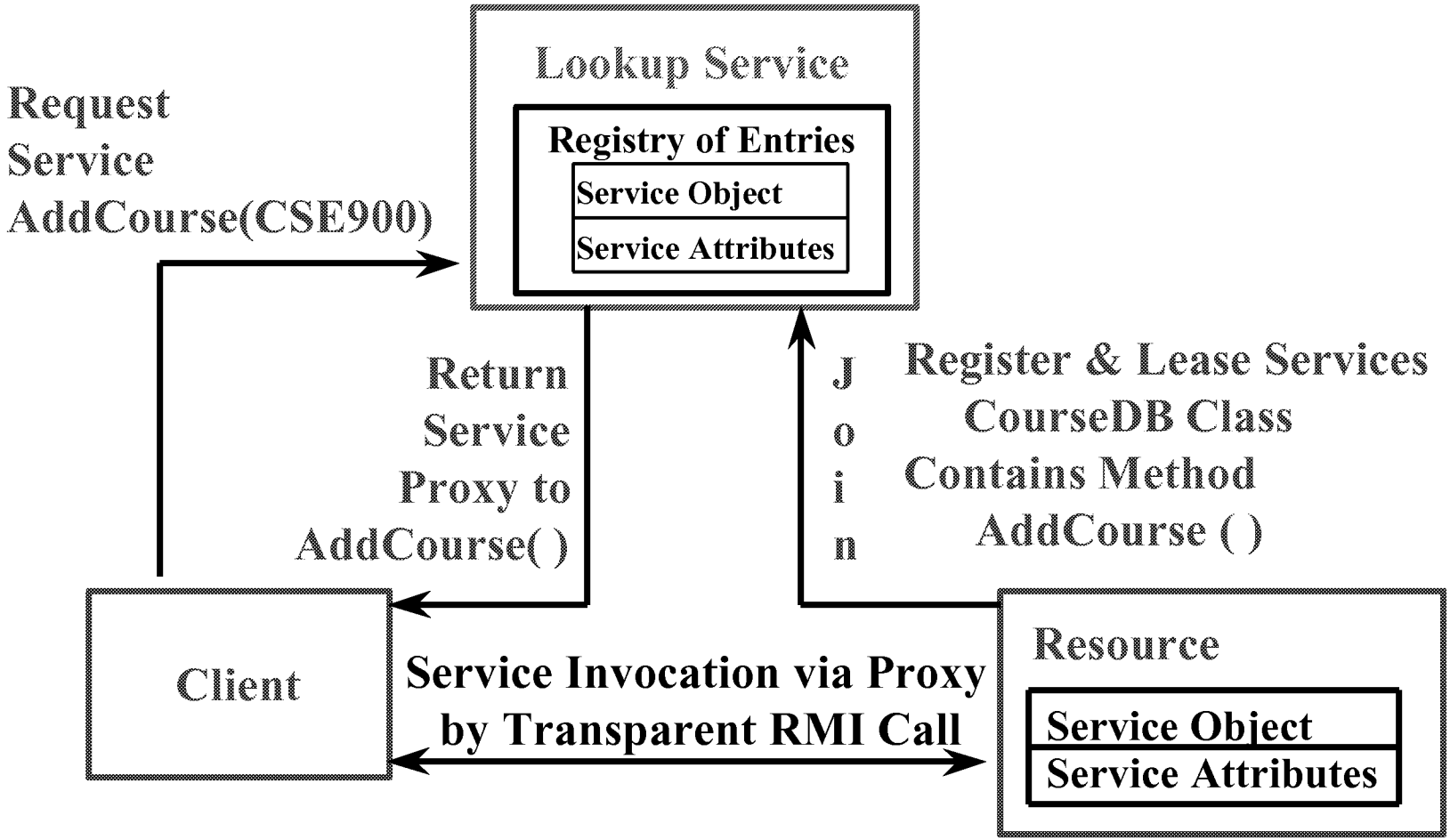


Sun's JINI Technology
Overall Computing Architecture and JINI.



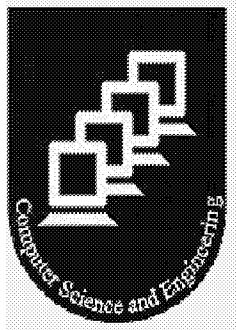
UCONN

Sun's JINI Technology
Join, Lookup, and Service Invocation



1. Client Invokes AddCourse(CSE900) on Resource
2. Resource Returns Status of Invocation

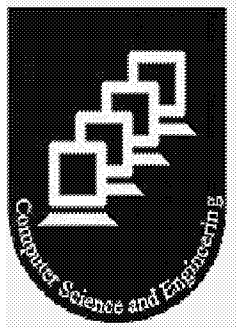
UCONN



Experimental Prototyping Effort Goals and Objectives

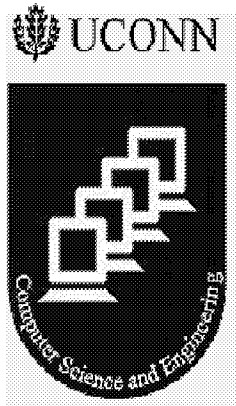
- High Availability of Services and Data
- Volatility of Resources, Clients, and Network
- Clients Rely on Lookup Service to Locate and Execute Services Against Replicated Resources
  - Databases Replicated in Multiple Workstations
  - Redundant Services Available if Failure
  - "Minutes-Off" Allowed - Sync Over Time
  - No Lost Updates During Modification Process
- **Characteristics of Enterprise Applications**
  - **Movement of Clients/Reconfigure Networks**
  - **Need for Data Availability on Demand**
  - **Receive "Current" Data if Client/Resource Plugs Back In**

UCONN



Experimental Prototypes Rapid, Incremental Design/Development

- First Prototype: Explore JINI and Develop Baseline University Application: Single Computer
- Second Prototype:Client and Services Spread to Two Computers: Lookup with Database Services
- Third Prototype: Extend Second to Multiple Clients and Three Computers
- Fourth and Fifth Prototypes
  - Single Client, Three Replicated Databases
  - **Testing of Replica Failures on Application**
  - Fifth: Multiple Clients/**Simultaneous Updates**
- **Sixth Prototype: Extends Fifth PT with Pre-Lookup Services for Locking During Updates**
- Six Prototypes in Six Weeks by 2 Grad Students

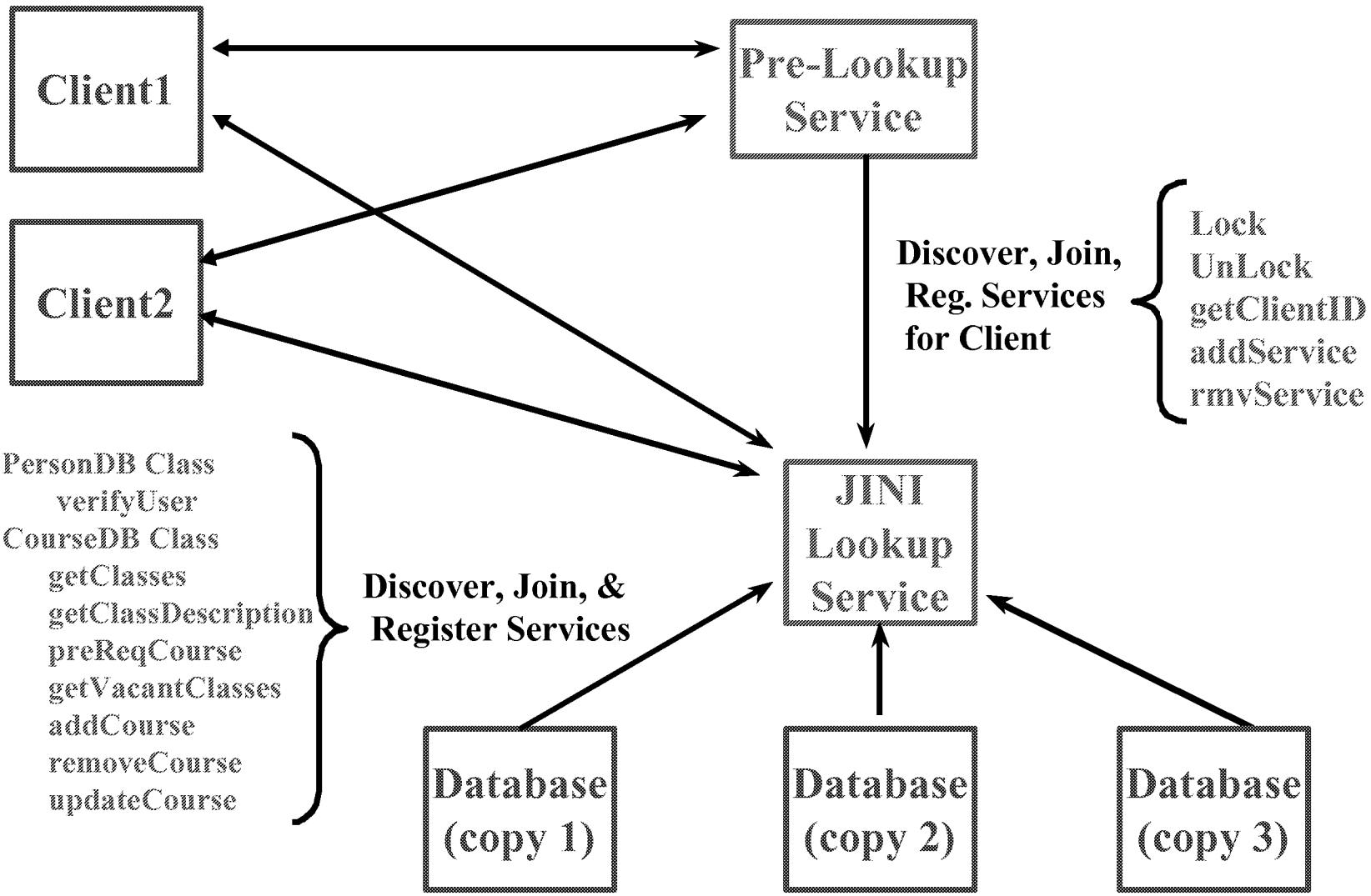


Prototype Six

- Functionality:
  - Incorporate Pre-Lookup Service that Insures Only One Client Updates Replicas
  - Use a Exclusive Write/Simultaneous Reads Protocol
  - Client Interacts with Pre-Lookup Service to "Request" Locks
  - Client Then Asks Lookup Service for Services
  - Client Receives and Updates All Replicas
- Purpose:
  - Bootstrap: Use JINI to Solve Update Problem
  - Eliminate Consistency Problems Across Replicas by Locking Databases During Update

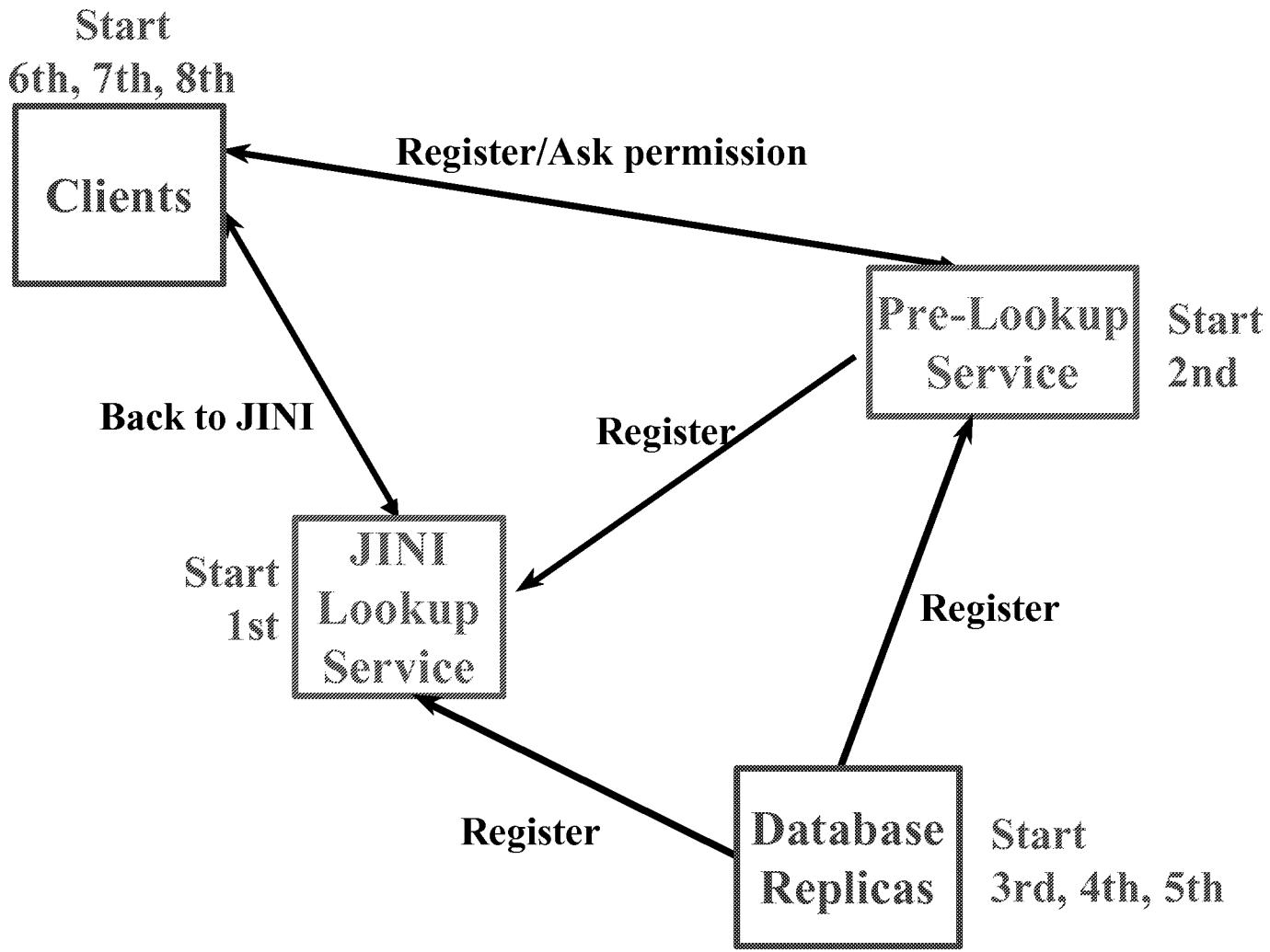
UCONN

Services in Prototype Six

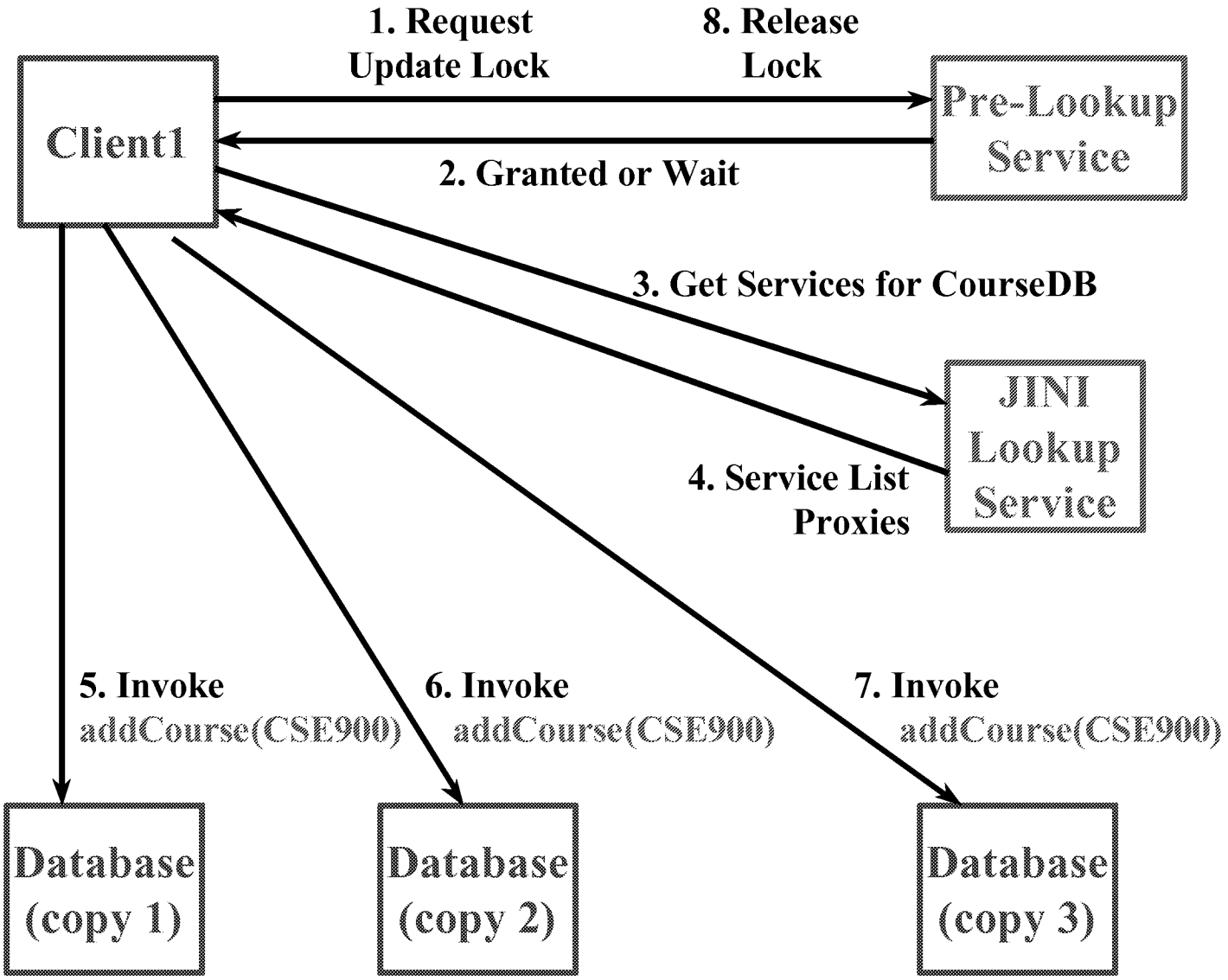


UCONN

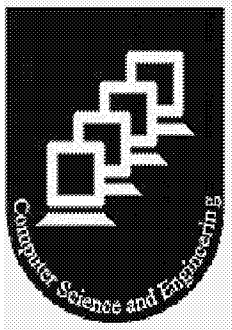
Order to Start Application



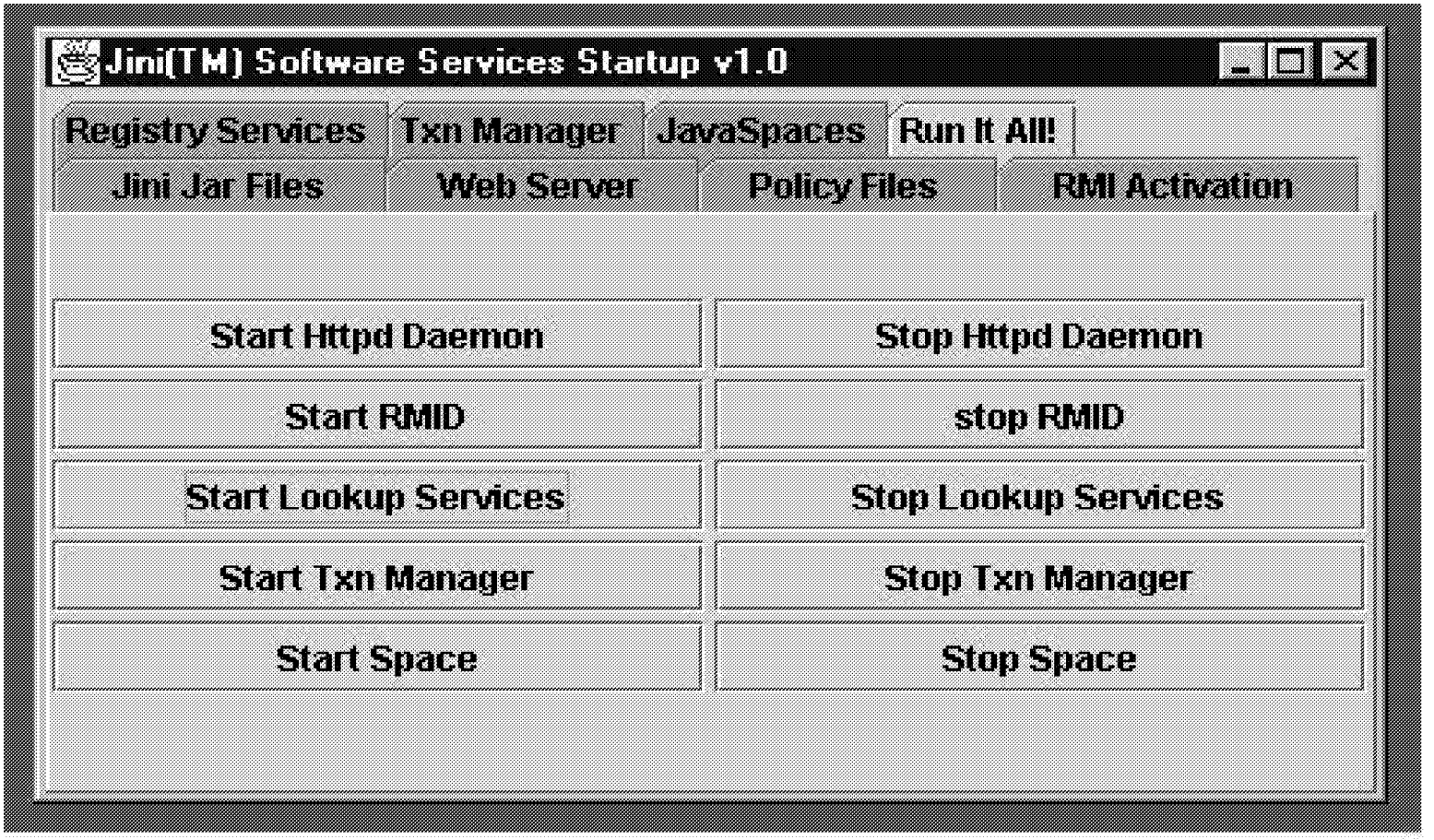
Execution Process in Prototype Six



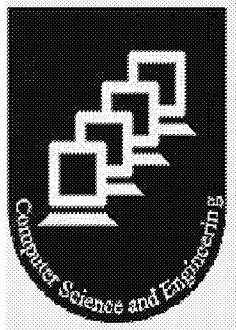
UCONN



Services GUI



UCONN



Pre-Lookup when Two Replicated Database Resources Register

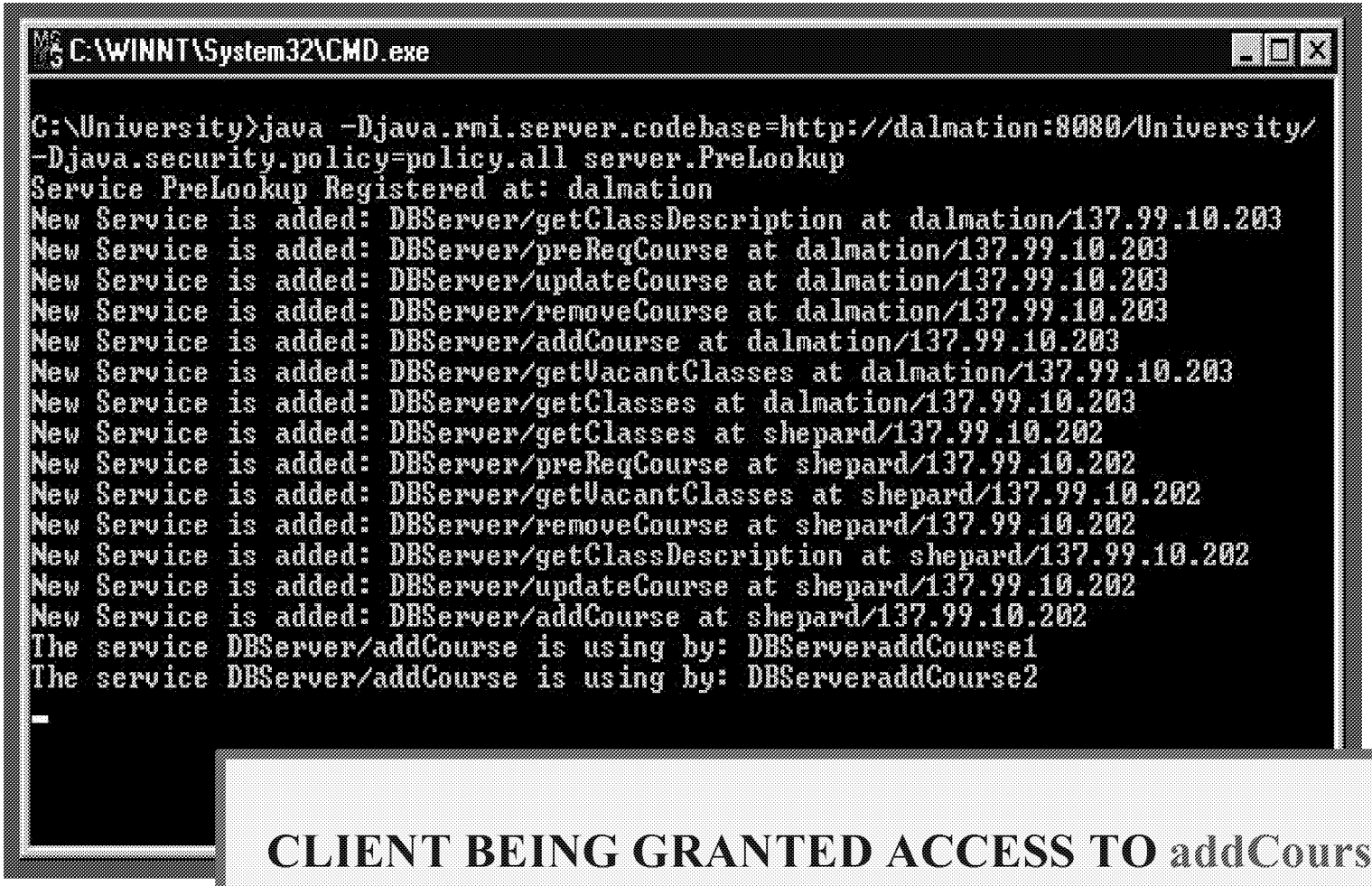
```
C:\WINNT\System32\CMD.exe

C:\University>java -Djava.rmi.server.codebase=http://dalmation:8080/University/
-Djava.security.policy=policy.all server.PreLookup
Service PreLookup Registered at: dalmation
New Service is added: DBServer/getClassDescription at dalmation/137.99.10.203
New Service is added: DBServer/preReqCourse at dalmation/137.99.10.203
New Service is added: DBServer/updateCourse at dalmation/137.99.10.203
New Service is added: DBServer/removeCourse at dalmation/137.99.10.203
New Service is added: DBServer/addCourse at dalmation/137.99.10.203
New Service is added: DBServer/getVacantClasses at dalmation/137.99.10.203
New Service is added: DBServer/getClasses at dalmation/137.99.10.203
New Service is added: DBServer/getClasses at shepard/137.99.10.202
New Service is added: DBServer/preReqCourse at shepard/137.99.10.202
New Service is added: DBServer/getVacantClasses at shepard/137.99.10.202
New Service is added: DBServer/removeCourse at shepard/137.99.10.202
New Service is added: DBServer/getClassDescription at shepard/137.99.10.202
New Service is added: DBServer/updateCourse at shepard/137.99.10.202
New Service is added: DBServer/addCourse at shepard/137.99.10.202
```

TWO DATABASE RESOURCES ARE INTERACTING WITH THE PRE-LOOKUP TO …

Register Services from DALMATION & SHEPARD:
getClassDescription, preReqCourse, updateCourse, removeCourse, addCourse, getVacantClasses, getClasses

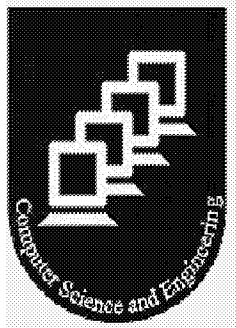
UCONN

Pre-Lookup when Client Invokes "addCourse" Service



CLIENT BEING GRANTED ACCESS TO addCourse SERVICE ON DALMATION & SHEPARD

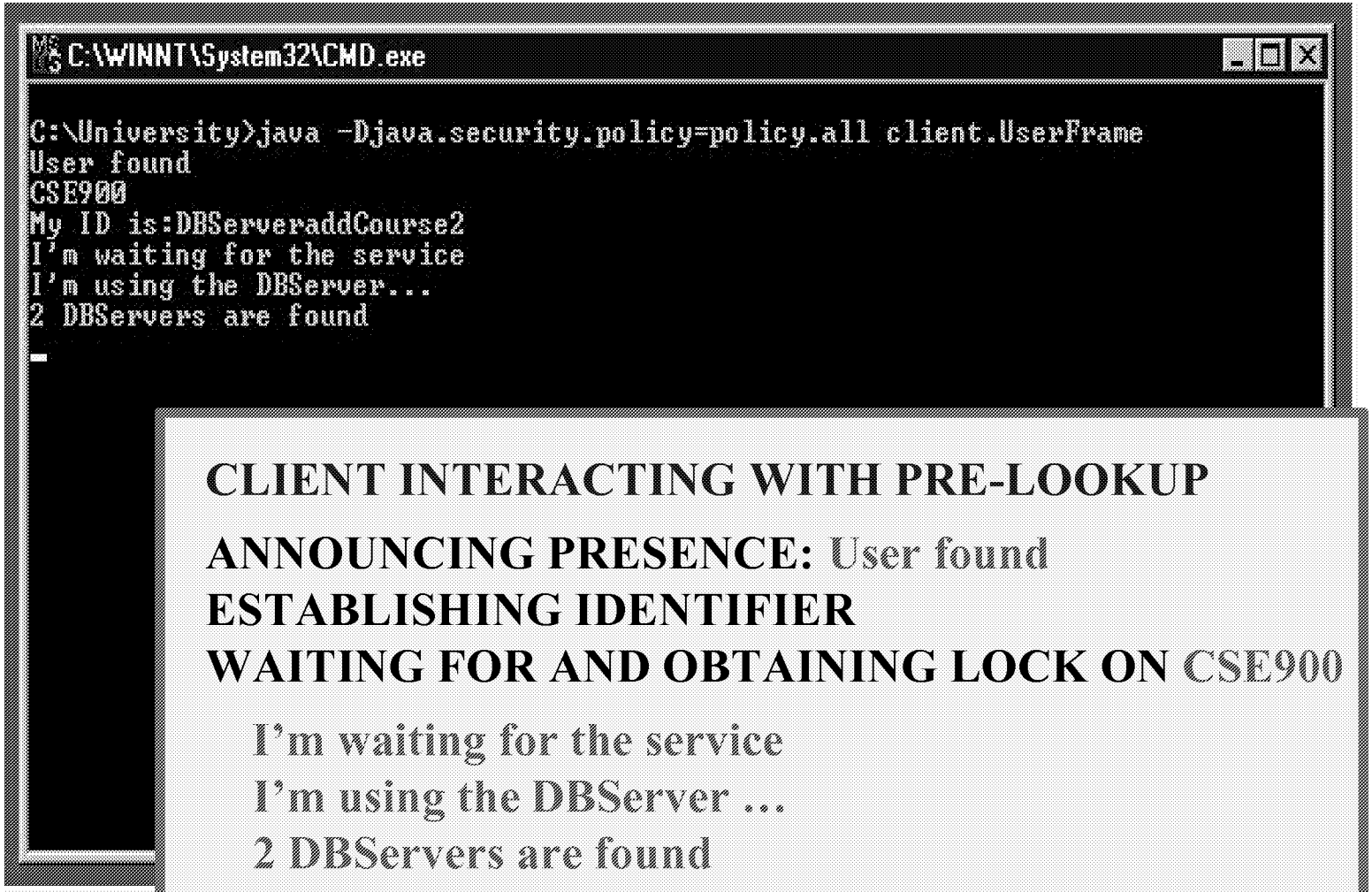
UCONN



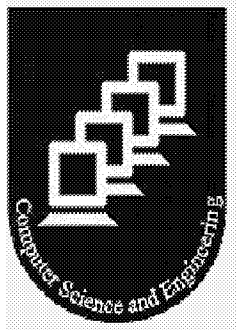
Client Passed Through Pre-Lookup

UCONN

Client Locked by Pre-Lookup

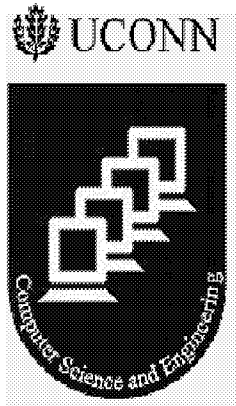


UCONN



Results from Prototype Six

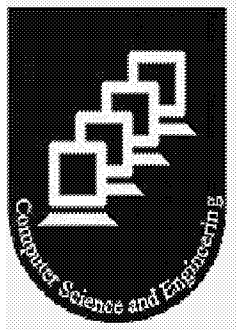
- Achievements:
  - **Any** Number of Clients Can Do the **Read** Operation on the Database
  - Only **One** Client Can **Modify** a Replicated Database Resource at a Given Point of Time
  - Pre-Lookup Service Can be Modified to Support other Locking Protocols
- Drawbacks:
  - **Can't Force Stakeholders to Consult Pre-Lookup Service - Rely on SW Practice**
  - No Redundant Lookup/Pre-Lookup Services
  - Failure of Lookup/Pre-Lookup - Total Restart
  - **No Logging and Recovery When Replicated Server Application Fails and is Restarted**



Experimental Prototyping Effort Future Work Possibilities

- **Verification of Write-Once-Run Anywhere**
  - Extend to Win95, 98, NT, Solaris, and Other Databases (Sybase, Oracle, …)
  - **CRITICAL for Comprehensive Evaluation of JINI's Readiness for 21st Century!**
- **Explore Other JINI Capabilities**
  - Transactions/Two-Phase Commit and JINI Security Model for Granting/Denial of Services
  - **Response, Consistency, Authorization**
- **Software Agents and JINI**
  - Ongoing Project this Semester
  - **Can Technologies Work Together?**
- **Others Include (JavaSpaces, Performance, …)**

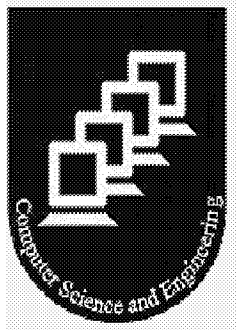
UCONN



Conclusions and Recommendations
Revisiting Goals of Research Effort

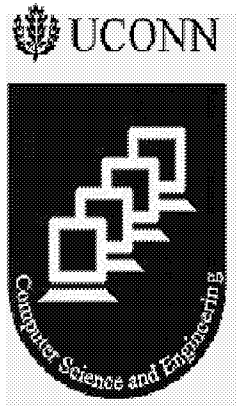
- Can JINI Support **Highly-Available Distributed Applications**?
  - PTs 4, 5 and 6 Demonstrate Ability to Architect Highly Available Solutions via JINI
- Can JINI Support an Environment with **Dynamic Clients and Replicated Services**?
  Will Clients Continue to Operate Effectively if **Replicated Databases Services Fail**?
  - Yes, PTs 4, 5, 6 All Support Starting and Stopping of Clients and Database Resources
- Can JINI be Utilized to **Maintain "minutes-off" Data Consistency** of Replicas?
  - PT 6 Superior Due to Pre-Lookup Service Guarantee of No Lost Updates

UCONN



Conclusions and Recommendations
Is JINI Ready for Prime Time???

- **Compatibility of JINI with Java**
  - Homogeneity of JINI and Java Unlike ORBs and IDL which are Heterogeneous
  - **But - Verify Across Heterogeneous HW/SW**
- **Track Record of Java and Sun**
  - Java's Increasing Dominance in Agents, E-Commerce, EJB, Data Mining, etc.
  - **But - Incompatibility of Java Releases and Danger of Evolving Technologies**
- **Understandability and Ease of Use of JINI**
  - 400 Hours Total for Obtaining Familiarity with JINI, Visual Café, JDBC/ODBC, 6 PTs, etc.
  - Faster Speed Up w.r.t. CORBA/ORB



Conclusions and Recommendations
Important Caveats

- **Future Work Clearly Indicates that More Effort is Required to Tilt Scales Toward JINI**
  - Work Extensive w.r.t. JINI Technologies
  - Potential Role of JavaSpaces
- **Experiences on Whole Positive**
  - Leasing Issues Need to be Explored
  - Impact of Registry Currency on Clients
- **Continued Evolution of JINI Specification**
  - Will JINI 1.1 be Compatible with JINI 1.0?
  - Java has Had Compatibility Problems in Past (Deprecated APIs)
- **Conclusion: JINI Great Promise as a Successful Technology in 21st Century!**

A Classification of Software Components Incompatibilities for COTS Integration

Daniil Yakimovich[⊗]
dyak@cs.umd.edu

[⊗]Experimental Software Engineering Group
Department of Computer Science
University of Maryland at College Park
A.V. Williams Building
College Park, MD 20742
USA

Guilherme H. Travassos[⊗,•]
travassos@cs.umd.edu

[•]Computer Science and System Engineering Department
COPPE
Federal University of Rio de Janeiro
C.P. 68511 - Ilha do Fundão
Rio de Janeiro – RJ – 21945-180
Brazil

Victor R. Basili[⊗,Θ]
basili@cs.umd.edu

[Θ]Fraunhofer Center - Maryland
3115 Ag/Life Sciences/Surge Bldg.
University of Maryland
College Park MD 20742
301-405-4770

ABSTRACT

Integration of software components into a system can be hindered by incompatibilities between the components and system. To predict the possible incompatibilities and the ways to overcome them during the integration activities, a classification of incompatibilities can be useful for software developers. This can be especially crucial for COTS-based software development, where a software system is being built out of potentially highly heterogeneous software components. The resulting system can have a complicated architecture due to the diversified nature of its components (e.g., a message-based system with object-oriented and procedural sub-systems), and the architectural incompatibilities of the COTS products must be overcome. Moreover, the functionality of the COTS software products must be taken into account during COTS integration. In this paper we present a classification of incompatibilities based on the properties of local component interactions. We believe that this classification can capture possible problems about software component integration in heterogeneous software systems, including architectural and functional issues.

1. INTRODUCTION.

Commercial-off-the-shelf software is developed by a third party and intended to be part of a new software system [McDermid, Talbert 97]. Usage of COTS products is growing, because developers hope that it will increase their systems quality and reduce development time. However, COTS based development implies specific problems (such as selection, integration, maintenance, and security) whose solutions can be illustrated by answering the following questions:

- How *to select* the most suitable COTS product in the market?
- How *to integrate* the COTS product into the new system?
- How *to maintain* a system that has components developed outside?
- How *safe* a COTS software product is?

These are just a few problems. In this paper we are going to discuss COTS integration and its impact on COTS selection. The importance of discussing COTS selection and integration show up when considering that COTS products are developed to be generic, however, being integrated into a system, they are used in a specific context with certain dependencies. The existence of mismatches between the COTS product being integrated and the system is possible due to their

different architectural assumptions and functional constraints. These mismatches must be overcome during integration and they have to be identified even earlier. Thus, a classification of mismatches or incompatibilities can be useful for COTS selection and integration.

There are some publications exploring integration architectural issues. For instance, [Gacek et al. 95], [Shaw 95], [Shaw, Clements 96] identify and classify architectural mismatches and styles. [Abd-Allah, Boehm 96] and [Gacek 98] deal with heterogeneous architectures. This is especially important for COTS development because a COTS-based software system can be built out of potentially highly diversified software components, which can result in a heterogeneous architecture (e.g., a message-based system with object-oriented and procedural sub-systems) for the software system. However, not just architectural mismatches must be considered for integrating COTS, but also the required functionality, non-functional constraints, and software developers expertise level.

A COTS product can have gaps in required functionality, it can have incompatible interfaces, different architectural assumptions, and it can conflict with other system components. Selecting suitable COTS products for a project can require finding a trade-off between different mismatches depending on the organization's development capabilities. For example, if an organization has a strong expertise in a functional domain but little experience in coping with architectural problems it can consider acquiring COTS products with less required functionality but with few architectural mismatches. On the contrary, if an organization is more experienced in architectures than in the domain it should select COTS products with as much functionality as possible, although there can be considerable architectural problems. The right selection can minimize the integration effort.

Therefore in this work we propose a general classification of possible types of mismatches between COTS products and software systems, which includes architectural, functional, non-functional, and other issues. We present a classification of incompatibilities based on the properties of local component interactions. We believe that this classification captures possible problems about software component integration in heterogeneous software systems. We expect that the incompatibility classification can help to estimate the effort (cost) of the integration of the COTS products prior to deciding about using a specific one. By utilizing it, software developers can decide about a COTS product early in the software process, anticipating the possible integration risks.

This paper has four sections including this introduction. Section 2 deals with the interactions and how such concepts can be explored to identify incompatibilities. The third section explores the whole model, showing which types of incompatibilities software developers should look for. Also, a short example of using such a scheme is presented. Section 4 concludes this discussion and shows some on going works regarding estimation of cost for COTS integration.

2. INTER-COMPONENT INTERACTIONS AND CLASSIFICATION.

The incompatibilities, for the context of this work, are essentially failures of components' interactions, so finding and classifying these interactions will help to find and classify the incompatibilities. We consider three aspects of inter-component interactions and incompatibilities: type of interacting component, layer (syntax or semantic-pragmatic), and number of components participating in the interaction.

First, the components interact with other system components, and with the system environment. System components can be either software or hardware (excluding everything related to the environment, such as CPU and memory, but including devices directly controlled by the system, such as on-board devices) that are used by the software system. The environment can be of the development phase, which includes compilers, debuggers, and other development tools, or it can be the environment of the target system, which includes Operating Systems, virtual machines (such as Java), interpreters (such as Basic), and other applications and utilities used by the target system. The parts of both environments can also be considered components. Figure 1 shows the different perspectives that can be used to classify these software component interactions.

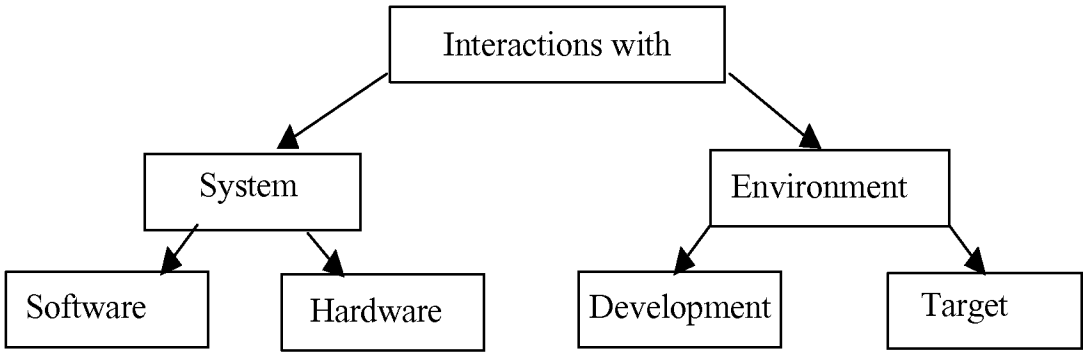


Figure 1. Interactions of software components.

Then two main layers can be differentiated in the inter-component interactions:

- **Syntax**, defines the representation of the syntax rules of the interaction, e.g., the name of invoked function; the names, types, and the order of the parameters or data fields in the message, etc. For instance, float SQRT(float x) represents a C notation for a function called "SQRT" returning a real result and with one argument, a real number x.

- **Semantic-pragmatic**, defines the functional (semantic and pragmatic) specifications of the interaction, i.e., what functionality is performed by the component, e.g., invoking the function "SQRT" calculates the square root of its only argument and returns it to the caller. However, in this work we do not consider semantic and pragmatic issues separately.

Finally, an incompatibility can occur in an interaction involving a certain number of participating components. A syntax incompatibility can occur because of syntactic difference between two components, but a semantic-pragmatic incompatibility can be caused either by just one component, two mismatching components, or three or more conflicting components. Thus, incompatibilities of the semantic-pragmatic layer can be classified according to the exact number of components that caused the interaction to fail. Therefore, the following types of semantic-pragmatic incompatibilities can be considered:

- **1-order semantic-pragmatic incompatibility**, or **an internal problem**, if a component alone has an incompatibility disregarding the components it is interacting with. It means that the component either does not have required functionality (not matching the requirements) or its invocation can cause a failure (an internal fault).

- **2-order semantic-pragmatic incompatibility**, or **a mismatch**, if an incompatibility is caused by interaction of two components. Both components may not have 1-order incompatibilities and can work correctly in other contexts. For example, a procedure that calculates the square root of a real number receives a negative argument from a caller that supposes that this is a valid output.

- **N-order semantic-pragmatic incompatibility**, or **a conflict**, if an incompatibility is caused by interactions of several components. There may not be semantic-pragmatic 1-order and 2-order incompatibilities for these components, but their cumulative interaction can cause a failure. For example, several processes together require more memory than the available amount, although each of them can be satisfied independently, so there is an n-order incompatibility on the semantic-pragmatic layer in interactions with the target platform.

According to the assumptions above, syntactical and semantic-pragmatic incompatibilities can occur in the system and environment dimensions. Table 1 captures this classification, where the cells are described below.

Type of component	*System*		*Environment*	
Type of incompatibility	Software	Hardware	Development	Target
Syntax	1.1	2.1	3.1	4.1
Semantic-pragmatic 1-order	1.2a	2.2a	3.2a	4.2a
Semantic-pragmatic 2-order	1.2b	2.2b	3.2b	4.2b
Semantic-pragmatic n-order	1.2c	2.2c	3.2c	4.2c

Table 1. Interactions incompatibilities.

1. Interactions with software

1.1. Syntax:

Three different types of syntax incompatibilities can be described here. Although there is only one cell capturing the idea of syntax issue for software in Table 1, its contents allows the identification of differences/incompatibilities regarding:

- Information flow, e.g., control instead of data.
- Binding: static, dynamic compile-time, dynamic run-time, topological, etc. As the result a component can not find another one.
- Interface protocol: different number of parameters or data fields, or different types of parameters or data fields.

1.2. Semantic-pragmatic:

1.2.a. 1-order: internal problem. These incompatibilities appear when the COTS product does not match the required functionality (e.g. it does not perform a required function), or due to its poor quality it still does not work properly (an internal fault). On the other hand, it can be other software that is solely responsible for the failure of interaction with the COTS product.

1.2.b. 2-order: different assumptions between two components, including the synchronization issue. These incompatibilities are products of a mismatch between the COTS product and other components surrounding it. Even when two components have correct functionality they can fail to work together due to some differences. (e.g., one object uses metric units, but another one uses inches, therefore the result can hardly be correct; another example is a mismatch between an asynchronous and a synchronous component).

1.2.c. N-order: a conflict between several software components. Even when the COTS product works correctly itself and correctly interacts with other components, some incompatibilities can appear as the result of a combined interaction with several other software components. (e.g., an object that controls rotation of a spacecraft receives the command for rotating on n degrees from a commanding object, but occasionally there is another commanding object, which sends the same command at the same time, in the system. Every single interaction is correct, but the spacecraft rotates twice as fast as it should do.)

2. Interactions with hardware

2.1. Syntax:

Different type of protocol. A software component can not work with a piece of hardware, because they assume different protocols (e.g. TCP/IP and Decnet or different port numbers).

2.2. Semantic-pragmatic

2.2.a. 1-order: wrong functionality of hardware or the COTS component. A hardware component does not work correctly (e.g. a printer does not support the Cyrillic alphabet), or the COTS component causes a failure.

2.2.b. 2-order: different assumptions between software and hardware. An interaction between software and hardware components does not work correctly (e.g., a program tries to print a Cyrillic text, but the printer has a different coding for the Cyrillic alphabet, therefore the output will be unintelligible).

2.2.c. N-order: a conflict between several software components over hardware. An interaction among several software components and a hardware component does not work correctly (e.g., several applications simultaneously accessing a single printer).

3. Interactions with the Development Environment

3.1. Syntax:

Different components' representation. The environment does not understand the packaging of a software component (e.g., a C program can not be compiled by a Fortran compiler).

3.2. Semantic-pragmatic:

3.2.a. 1-order: wrong functionality of the environment or the COTS component. The environment does not work properly (e.g., a defect in the compiler version), or the component has an error (e.g., a program can not be compiled because of a syntax error in it).

3.2.b. 2-order: different assumptions between the software component and the environment. A software component can not interact with the environment (e.g., a program is written in an old dialect of the language and can not be compiled by a newer compiler).

3.2.c. N-order: a conflict between several software components over the environment. An interaction among several software components and the development environment causes an incompatibility (e.g. two or more C modules can not be compiled or linked together because of a name collision).

4. Interactions with the target environment

4.1. Syntax:

Platform type. The environment does not understand the packaging of a software

component (e.g., a program uses another OS, or an interpreter can not run a program written in another language).

4.2. Semantic-pragmatic:

4.2.a. 1-order: wrong functionality of the environment or the COTS component. The environment does not work properly (e.g., the OS crashes), or the component has an error (e.g., a memory violation in a program).

4.2.b. 2-order: different assumptions between the software component and the environment. A software component does not interact with the environment correctly (e.g., a different version of the OS version performs some functions used by the component in a way other than expected by the component's developers).

4.2.c. n-order: a conflict between software components over the environment, including the control issue. An interaction among several software components and the environment causes an incompatibility (e.g. a conflict between two object-oriented frameworks in a one-process program for the control flow [Sparks et al. 96]).

3. TYPES OF INTEGRATION PROBLEMS.

Different incompatibilities have different solutions, but generally we can find five groups of related problems with the proper solution strategies. We assume that one type of incompatibilities can cause problems in different groups. For example, a syntax software incompatibility can cause different types of binding, which can require a special architectural solution for the whole system, or it can be just a different order of parameters, which can be overcome by a simple wrapper. Thus, we can differentiate the following groups of integration problems:

- **Functional**. All the 1-order semantic-pragmatic incompatibilities that are caused by missing or wrong functionality. Re-implementation or modification of faulty components can solve these problems.

- **Non-functional**. Some 1-order semantic-pragmatic incompatibilities can be caused by not matching to non-functional requirements, such as reliability, maintainability, efficiency, usability, etc. These problems are difficult to solve without reworking the component.

- **Architectural**. These issues constitute another class of problems and can cause changing the overall system's architecture, but the incompatibilities causing them are different. In this work we consider the following architectural assumptions of software components with their respective incompatibilities: packaging (syntax development and target environments), control (n-order semantic-pragmatic target environment), information flow (syntax software), binding (syntax software), synchronization (2-order semantic-pragmatic software) [Shaw 95], [Yakimovich et al. 99].

- **Conflicts**. Problems of this type are conflicts between components in the system (e.g., deadlocks). The related incompatibilities are n-order semantic-pragmatic software and hardware. The possible solutions can include changing the system's configuration without changing the overall architectural type (minor architectural changes, including monitoring components) and using glueware.

- **Interface**. These problems are incompatible interfaces between the components caused by some syntax and 2-order semantic-pragmatic software and hardware incompatibilities (other

than major architectural). The possible solution is glueware.

Another property of this high-level classification is that the classes of problems are specific to the particular development phases. Functional and non-functional issues require information on the project and COTS product functionality, which is available early in the requirements analysis phase. Architectural issues are dealt with during the design phase when the system's architecture is being designed. Conflicts and interface issues are addressed later in the design phase when the system's architecture and the component's interfaces are known.

Let us consider the following example to illustrate our approach; a 3D-graphics engine is being chosen for a real-time system. The system being developed imposes the following high-level requirements for the graphics engine:

Functionality: drawing 3-dimensional objects, including input and output 3D images from files.
Non-functional issues (portability): Mac.
Architectural issues (development platform): Ada 95.
Interfaces (example of a function): procedure Rect(x, y, w, h: Real); where (x,y) – the coordinates of the left bottom corner of the rectangle; w – its width; h – its height; output – drawing a rectangle. Other specifications, such as non-functional requirements, hardware requirements, possible conflicts, etc., are not considered in this example.

The possible candidate COTS products are OpenGL, QuickDraw3D, and DirectX [Thompson 96]. Matching them against the requirements gives the following data:

OpenGL:

> *Functionality*: the drawing functions are provided, input and output from files is not supported – 1-order semantic-pragmatic incompatibility.
> *Non-functional issues (portability):* Mac platform is supported.
> *Architectural issues (development platform):* an Ada implementation is available.
> *Interface:* procedure glRectf(x1:GLfloat; y1:Glfloat; x2:Glfloat; y2:GLfloat); where (x1,y1) – the coordinates of one vertex of the rectangle; (x2,y2) – the coordinates of the opposite vertex of the rectangle. There are a syntax incompatibility (different procedure names) and a 2-order semantic-pragmatic incompatibility (different interpretations of the arguments) with software components.

QuickDraw3D:

> *Functionality:* drawing provided, input and output from files is supported.
> *Non-functional issues (portability):* Mac platform is supported.
> *Architectural issues (packaging):* Ada 95 implementation is not available – 2-order semantic-pragmatic incompatibility with the development platform.
> *Interface:* it is not necessary to consider it, because it is expensive to use QuickDraw3D due to the different packaging.

DirectX:

> *Functionality*: drawing provided, input and output from files is supported.
> *Non-functional issues (portability):* Mac platform is not supported – 2-order semantic-pragmatic incompatibility with the target platform.

Architectural issues (packaging): Ada 95 implementation is not available – 2-order semantic-pragmatic incompatibility with the development platform.
Interface: it is not necessary to consider it, because it is extremely expensive to use DirectX due to the different packaging and target platform.

The result of this comparison is that OpenGL is the best candidate, despite certain incompatibilities that can be overcome using glueware and re-implementation. Use of C-implemented QuickDraw3D would require changing the system's architecture. Use of DirectX would require porting it to Mac, which is hardly a real operation.

4. CONCLUSIONS AND ON-GOING WORKS.

In this paper we presented a classification of incompatibilities between software (including COTS) components and other parts of a software system. This classification is intended to find the possible problems, including functional, architectural, non-functional, conflict, and interface, when a COTS software component is being integrated into a system. We hope that the incompatibility classification and the effort estimation approach can be useful for software developers to evaluate and integrate COTS software.

We have given above a classification of possible incompatibilities between the software (COTS) and other system components. However, to select a COTS product, developers must also know the effort required for overcoming these incompatibilities. To estimate the integration effort developers have to answer the following sequence of questions:

- *What are the incompatibilities?* - What is the difference between the system's requirements and the COTS products. This difference can be found using approaches, such as the comprehensive reuse model [Basili, Rombach 91].

- *How are they to be overcome?* - What integration strategies can be used by the developers to integrate the COTS software products (e.g., re-implementation, glueware, changes of architecture).

- *What is the amount of integration work?* - This is a quantitative estimation of the two items above; how much work is to be done to fill a certain gap.

- *What is the productivity (skill) of the developers for the applied integration strategy?* - This reflects the skill of the developers with respect to particular integration tasks. The higher it is, the faster they can perform the same amount of work. It can be possible to define techniques in different strategies, for example, re-implementation using object-oriented, procedural, or another paradigm. Specifying techniques within the strategies will demand more data about the organization, but on the other hand, the analysis will be more fine-tuned.

- *What is the effort required for overcoming a particular incompatibility between a COTS product and the system?* – This is obtained from the previous two items by dividing the amount of work by the productivity.

- *What is the total effort required for integrating a COTS product?* – This is the sum of the efforts required for resolving all the incompatibilities between the COTS product and the system.

Essentially, this is a bottom-up effort estimation model: each of the COTS product components is analyzed with respect to all its possible interactions with system to be integrated in. If an incompatibility is found the effort to overcome is estimated based on the amount of integration work and the productivity of organization for this type of work. The overall integration cost is the sum of overcoming all the incompatibilities between the COTS product's components and the system. However, to develop this COTS evaluation approach we must find effective ways to measure the productivity and the gap between the requirements and the system being developed.

As a research work, a process model for COTS selection, evaluation, and integration is being defined incorporating the ideas showed in this paper. Some experiments have been planned to empirically validate such a model. The results of these experiments, and the whole model, will be described in future publications.

REFERENCES:

[Abd-Allah, Boehm 96] Abd-Allah, A., Boehm, B., "Models for composing heterogeneous software architectures", USC Technical report: USC-CSE-96-505, University of South California, Los Angeles, August 1996.

[Basili, Rombach 91] Basili, V., Rombach, H., "Support for comprehensive reuse", Software Engineering Journal, September 1991, pp. 303-316.

[Gacek 98] Gacek, C., "Detecting Architectural Mismatches During Systems Composition," Doctoral Dissertation, Center for Software Engineering, University of Southern California, Los Angeles, CA 90089, December 1998.

[Gacek et al. 95] Gacek, C., Abd-Allah, A., Clark, B., Boehm, B., "On the definition of software system architecture", in the proceedings of the First International Workshop on architectures for software systems – in cooperation with the 17th international conference on software engineering, Seattle, WA, 24-25 April 1995, pp. 85-95.

[Garlan et al. 95] Garlan, D., Allen, R., Ockerbloom, J., "Architectural Mismatch or Why it's hard to build systems out of existing parts", Proceedings of International Conference on Software Engineering, 1995, Seattle, WA, USA, pp. 179 – 185.

[McDermid, Talbert, 97] McDermid, J., Talbert, N., "The Cost of COTS" (interview), Computer, June 1997, pp. 46-52.

[Shaw 95] Shaw, M., Architectural Issues in Software Reuse: It's Not Just the Functionality, It's Packaging, Proceedings of the Symposium on Software Reusability, 1995, Seattle, WA, USA, pp. 3-6.

[Shaw, Clements, 96] Shaw, M., Clements, P., "A field guide to boxology: preliminary classification of architectural styles for software systems", http://www.cs.cmu.edu/afs/cs.cmu.edu/project/vit/www/paper_abstracts/Boxology.html, Computer Science Department and Software Engineering Institute, Carnegie Mellon University, 1996.

[Sparks et al. 96] Sparks, S., Benner, K., Faris, C., "Managing Object-Oriented Framework Reuse", IEEE Computer, September 1996, pp. 52-61.

[Thompson 96] Thompson, T., "Must-See 3-D Engines", Byte, June 1996, pp. 137-144.

[Yakimovich et al. 99] Yakimovich, D., Bieman, J.M., Basili, V.R., "Software architecture classi fication for estimating the cost of COTS integration", Proceedings of the 21st International Conference on Software Engineering, Los Angeles, USA, 1999, pp. 296 –302.

A classification of software components incompatibilities for COTS integration

Daniil Yakimovich
University of Maryland

Guilherme H. Travassos
University of Maryland / Federal University of Rio de Janeiro

Victor R. Basili
University of Maryland / Fraunhofer Center for Experimental Software Engineering

Outline

- The COTS usage and problems
- Effort estimation model for integration
- **Incompatibility classification**
- Examples of incompatibilities
- Classes of issues and integration strategies
- Example of applying the classification for COTS selection
- Conclusions and future work

COTS usage

COTS (commercial-of-the-shelf) - commercially available, can be used as a part of a new system

Potential benefits:

Quality increase, Effort reduction

Issues:

Selection, Integration, Security, etc.

The integration issues

COTS products are developed in certain context but used in different projects.

Therefore, there exist incompatibilities:

- Functional
- Architectural
- Other

The incompatibilities must be predicted and overcome.

Selection and Integration

Selection must take in to account the integration cost

An algorithm for integration effort estimation:

- Find the incompatibilities between the COTS products and the system
- Estimate the effort of their overcoming
- Sum up these efforts for all incompatibilities and all COTS components

Effort estimation

Integration Effort (staff-hour) =

Amount of Work / Productivity

Amount of Work - the difference between the requirements and the reused COTS (LOC) - depends on the *incompatibilities*

Productivity depends on the organization (LOC/staff-hour)

The approach

The incompatibilities are failures of interactions between components.

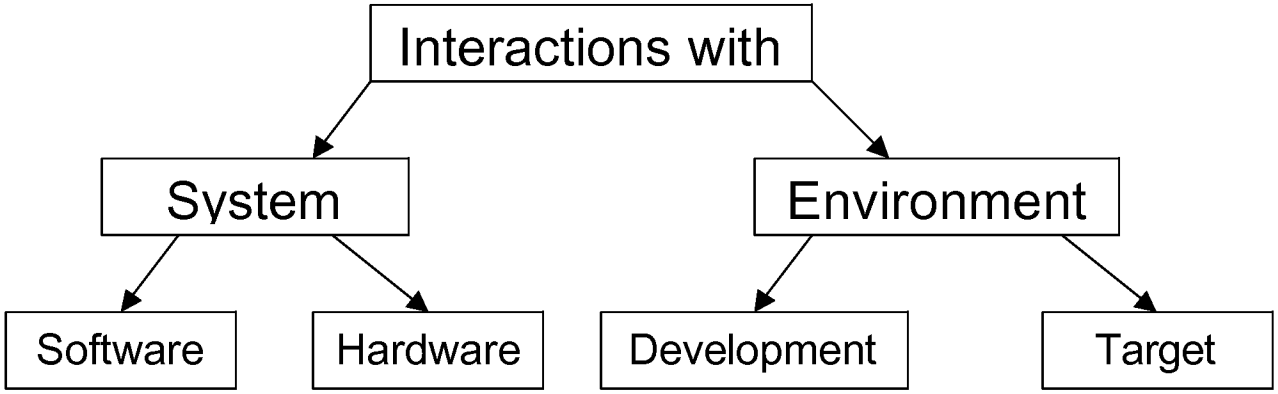
Thus, classifying the interactions can help to classify the incompatibilities.

Three interaction aspects:

- *type* of interacting components
- *layer* (syntax or semantic-pragmatic)
- *number* of interacting components

Types of interacting components

A software component can interact with:



Interaction layers

- *Syntax* - how the syntax rules of the interaction are defined,

  e.g. float SQRT(float x)
- *Semantic-pragmatic* - how the functionality of the interaction is defined,

  e.g. function SQRT calculates the square root of a non-negative number x

Number of interacting components

Semantic-pragmatic layer depends on the number of interacting components

- *1-order* - component itself (*internal fault*)
- *2-order* - interaction of two components (*mismatch*)
- *n-order* - interaction of three and more components (*conflict*)

Examples of incompatibilities

- *Syntax*: different names, different binding, different languages
- *Semantic-pragmatic*
  - *1-order*: wrong functionality
  - *2-order*: metric units vs. inches, different synchronization, different encodings
  - *n-order*: data sharing violations, simultaneous access, name space collisions

Classes of issues

Type of component	*System*		*Environment*	
Type of incompatibility	Software	Hardware	Development	Target
Syntax	**I,A**	I	I(?),A	I(?),A
Semantic-pragmatic 1-order	F,NF	F,NF	F,NF	F,NF
Semantic-pragmatic 2-order	I,A	I	I(?),A	I(?),A
Semantic-pragmatic n-order	C	C	C(?),A	C(?),A

F – functional problem
NF – non-functional problem
A – architectural problem

C – conflict problem
I – interface problem

Integration strategies

- *Functional* - re-implementation
- *Non-functional* - discarding the COTS?
- *Architectural* - changes in the architectural style
- *Conflicts* - changes in the architecture
- *Interfaces* - glueware

Example

System requirements

Functionality: drawing 3D objects, input and output for 3D images from files

Non-functional issues (portability): Mac

Architectural issues (packaging): Ada

Interface: procedure Rectangle(x,y,w,h:real);

COTS: QuickDraw3D

Functionality: all functions are provided

Non-functional issues: Mac is supported

Architectural issues: Ada implementation is **not** available

COTS: OpenGL

Functionality: input and output for 3D images from files is **not** provided

Non-functional issues: Mac is supported

Architectural issues: Ada implementation is available

Interface: procedure glRectf(x1,y1,x2,y2:Glfloat);

COTS: DirectX

Functionality: all functions are provided

Non-functional issues: Mac is **not** supported

Architectural issues: Ada implementation is **not** available

Conclusions

- We presented a classification of incompatibilities between software components and other parts of a system
- We provided some insight on the issue of COTS integration with respect to finding and overcoming incompatibilities
- We outlined an approach for integration effort estimation

Ongoing work

- Validating this classification in practice (NASA/SEL)

- Improving the proposed COTS selection and integration approach based on this incompatibility classification

Appendix A – Workshop Attendees

Abraham, William L	Rowan University	willeonabe@msn.com
Abshire, Gerald	Computer Sciences Corp	gabshire@cscmail.csc.com
Allen, Theodosia	IRS	theodosia.l.allen@m1.irs.gov
Amacher, Aaron G	ICE	agamache@aol.com
Ammari, Habib	WV University	ammari@csee.wvu,edu
Anderson, Allan	Hughes Network Systems	aanderson@hns.com
Anderson, Frances E	IIT Research Institute	fanderson@iitri.org
Arrighi, Heather	Raytheon ITSS	harrighi@pop600.gsfc.nasa.gov
Aydinlioglu, Baris	University of Maryland	baris@cs.umd.edu
Ayers, Everett	Ayers Associates	[No Email Address]
Bae, Youn Y	NASA/GSFC	youn.y.bae@gsfc.nasa.gov
Ballard, Ben	JHU/APL	ben.ballard@jhuapl.edu
Basili, Victor R	University of Maryland	basili@cs.umd.edu
Batluck, Georgiann	NASA/GSFC	batluck@tiffy.gsfc.nasa.gov
Becker, Greg	QSS	gbecker@pop200.gsfc.nasa.gov
Beifeld, David	Unisys Corp	dbeifeld@pop300.gsfc.nasa.gov
Bergmann, Seth	Rowan University	bergmann@rowan.edu
Blazek, Ronald P	AlliedSignal	blazekr@rmtva.grbmd-p01
Blazy, Louis J	NASA IV&V	louis.blazy@ivv.nasa.gov
Blundell, Paul	Programming Research	paul_blundell@prqa.co.uk
Bobo, Jack A	ICE	[No Email Address]
Boger, Jacqueline	CSC	jboger@cscmail.csc.com
Bonas, Rachael	Howard University	rbonas@hotmail.com
Brandenburg, Wilber	NASA/GSFC	bbranden@pop700.gsfc.nasa.gov
Brown, Patrick	MITRETEK Corp	pbrown@mitretek.org
Budlong, Faye C	C.S. Draper Lab	budlong@draper.com
Carver, Jeff	University of Maryland	[No Email Address]
Caulfield, Margaret I	NASA/GSFC	margaret.caulfield@gsfc.nasa.gov
Centa, Alan	NASA/GSFC	acenta@pop500.gsfc.nasa.gov
Chambliss, Sandra H	AlliedSignal Aerospace	sandra.chambliss-nontrw@trw.com
Charron, Daniel	CAE Electronics Ltd	[No Email Address]
Chien, I-Ming Annie	Computer Sciences Corp	ichien@cscmail.csc.com
Chiverella, Ron	Highmark, Inc	ronald.chiverella@highmark.com
Choates-Workman, Mary	SOLIPSYS	mary.workman@solipsys.com
Chu, Martha	JHU/APL	martha.chu@jhuapl.edu
Chu, Richard	Lockheed Martin Corp	rchu@v2pop.hst.gsfc.nasa.gov
Chung, John	Computer Sciences Corp	jchung@csc.com
Cingel, Keith	ARINC	kac@arinc.com
Coleman Dangle, K	Fraunhofer Maryland	kdangle@fraunhofer.org

Colyandro, Anthony T	NASA/GSFC	anthony@dao.gsc.nasa.gov
Conradi, Reidar	NTNU	conradi@idi.ntnu.no
Cook, John F	NASA/GSFC	jcook@pop500.gsfc.nasa.gov
Cowan, James L	DISA	cowanj@ncr.disa.mil
Crispell, Michele	SATC	mcrispel@pop300.gsfc.nasa.gov
Curto, Paul	NASA/HQ	pcurto@hq.nasa.gov
Daddio, Ernest	NOAA	ernest.daddio@noaa.gov
De Fainchtein, R	Raytheon	rfaincht@gsfc.nasa.gov
Decker, William J	Computer Sciences Corp	wdecker@csc.com
Delguercio, Vincent	FAA Technical Center	vincent.delguercio@tc.faa.gov
Demurjian, Steve A	University Connecticut	steve@engr.uconn.edu
Derr, Patricia K	Dept. of Commerce	pderr@doc.gov
Derrick, Deborah	Computer Sciences Corp	dderrick@cscmail.csc.com
Dhama, Harpal	The MITRE Corp	dhama@mitre.org
Djidji, Domou F	Howard University	djicars@yahoo.com
Doyle, Richard J	Jet Propulsion Lab	Richard.j.doyle@jpl.nasa.gov
Drake, Anthony	Raytheon ITSS	adrake@daac.gsfc.nasa.gov
Duo, Jing	General Sciences Corp	jguo@dao.gsfc.nasa.gov
Dwyer, Al	Raytheon ITSS	adwyer@pop3.stx.com
Eickelmann, Nancy	NASA IV&V Facility	nancy.eickelmann@ivv.nasa.gov
Elderkin, Renee	Computer Sciences Corp	relderki@cscmail.csc.com
Ellis, Walter J	Software Process & Metrics	waltelli@erols.com
Evangelist, Michael	National Science Foundation	mevangel@nsf.gov
Fakory, Reza	Computer Sciences Corp	rfakory@v2pop.hst.nasa.gov
Ferrell, Tom	SAIC	[No Email Address]
Ferrell, Uma D	Reliable Software Tech	uferrell@rstcomp.com
Florence, Alfred W	The MITRE Corp	florence@mitre.org
Frey, Michael	Fraunhofer Center-Maryland	mfrey@fc.md.umd.edu
Fulmer, Dan	[No Organization Registered]	[No Email Address]
Futcher, Joseph M	Naval Surface Warfare Center	jfutcher@nswc.navy.mil
Gantzer, Donald J	TRW	don.ctr.gantzer@faa.gov
Garrahan, Jim	Computer Sciences Corp	jgarraha@csc.com
Gaston, Ralph	Computer Sciences Corp	rdgaston@erols.com
Gentle, Karen L	AlliedSignal T S	gentlek@atsc.allied.com
Gilbert, Jacqueline	FBI	[No Email Address]
Girma, Antenem	Howard University	antuye@aol.com
Glazener, Steve	L3Com	[No Email Address]
Godfrey, Pat	McCabe & Associates	[No Email Address]
Gomez, Sue	AlliedSignal T S	sue.gomez@alliedsignal.com
Gopalan, Venkat R	DynCorp	gopalav@dyncorp.com
Graham, Scott R	U.S. Air Force	scott.graham@afotec.af.mil

Green, Scott	NASA/GSFC	sgreen@pop500.gsfc.nasa.gov
Habetz, Marco	Fraunhofer Center-Maryland	mhabetz@fc-md.umd.edu
Halvorsen, Christian	NTNU	cph@idi.ntnu.no
Helm, James C	University Houston-ClearLake	helm@cl.uh.edu
Herndon, Thomas S	Computer Sciences Corp	therndon@csc.com
Hillelsohn, Michael	Software Performance Systems	hillelsohn@gosps.com
Hines, Tonjua	NASA/GSFC	thines@pop500.gsfc.nasa.gov
Houchens, Connie M	NASA/GSFC	chouchen@pop500.gsfc.nasa.gov
Houdek, Frank	Daimler Chrysler AG	frank.houdek@daimlerchrysler.com
Howlett, Alan	T.Rowe Price	ahowlett@troweprice.org
Hughes, Peter	NASA/GSF	phughes@pop500.gsfc.nasa.gov
Iona, Glenn	NASA/GSFC	giona@rattler.gsfc.nasa.gov
Jackson, Anthony	SATC	ajackso@pop300.gsfc.nasa.gov
Jamison, Donald	NASA/GSFC	djamison@pop500.gsfc.nasa.gov
Jeffery, Ross	University of New South Wales	rossj@cumulus.csd.unsw.oy.au
Jeletic, Jim	NASA/GSFC	jjeletic@pop500.gsfc.nasa.gov
Jeletic, Kellyann	NASA/GSFC	kjeletic@pop500.gsfc.nasa.gov
Jing, Yin	Computer Sciences Corp	yin.jing@cscgt.gsfc.nasa.gov
Jordano, Tony J	SAIC	anthony.j.jordano@saic.com
Kassebaum, Kass	Process & Change Management	kkass@erols.com
Kea, Howard E	NASA/GSFC	hekea@pop500.gsfc.nasa.gov
Kelley, Ken	Consultant	conoy@erols.com
Kelly, John C	Jet Propulsion Lab	john.c.kelly@jpl.nasa.gov
Kelly, Michael	Computer Sciences Corp	mkelly21@cscmail.csc.com
Kelly, Vernon	SAIC	kelly@saic.com
Kieckhefer, Ron	Computer Sciences Corp	rkieckhe@cscmail.csc.com
Kim, Yong-Mi	Q-Labs, Inc.	yong-mi.kim@q-labs.com
Koslosky, Anne Marie	NASA/GSFC	anne.koslosky@gsfc.nasa.gov
Kotov, Alexei	Software Process & Metrics	kotov@cse.ogi.edu
Kraft, Steve	NASA/GSFC	stkraft@pop500.gsfc.nasa.gov
Kuykendall, Frank	Jet Propulsion Lab	frank.kuykendall@jpl.nasa.gov
Landis, Linda C	Computer Sciences Corp	llandis@.csc.com
Lane, Martha	FBI	[No Email Address]
Leake, Steven	NASA/GSFC	stephen.leake@gsfc.nasa.gov
Lee, Anthony A	Marconi Systems Technologies	lee@tst.tracor.com
Lee, Michael H	NASA/GSFC	michael.h.lee@gsfc.nasa.gov
Lee, Roger A	Jet Propulsion Lab	roger.a.lee@jpl.nasa.gov
Legg, Jim	Raytheon	jim.legg@gsfc.nasa.gov
Li, Nelson	GST	[No Email Address]
Lin, Chi Y	Jet Propulsion Lab	chi.y.lin@jpl.nasa.gov
Lindvall, Mikael	Fraunhofer Center-Maryland	mlindvall@fraunhofer.org

Liu, Jean C	Computer Sciences Corp	jcliu@cscmail.csc.com
Lott, Christopher M	Telcordia Technologies	c.m.lott@ieee.org
Lubelczyk, Jeffrey T	NASA/GSFC	jlubelcz@pop500.gsfc.nasa.gov
Ludford, Joe	White Hart Associates	jludford@radix.net
Lyster, Peter M	University of Maryland	lys@dao.gsfc.nasa.gov
MacKenzie, Garth R	University of Maryland	gmackenz@umuc.edu
Major, Melissa L	Software Architects	major@software-architects.com
Marciniak, John J	Marciniak & Associates	jmarcin222@aol.com
Marjara, Amarjit	Cap Gemini	amarjara@hotmail.com
Maury, Jesse	Omitron, Inc	jesse.maury@omitron.com
McClinton, Arthur	MITRETEK Systems	art@mitretek.org
McCormick, Scott	TMC/Marada	smccormick@marada-corp.com
McDonald, James	AlliedSignal T S	mcdonaj@lskmpoo4.atsc.allied.com
McGarry, Frank E	Computer Sciences Corp	fmcgarry@csc.com
McGibbon, Thomas	DACS	tom.mcgibbon@ssc.de.ittind.com
McLay, Robert	University of Texas	[No Email Address]
Mendonca, Manoel G	University of Maryland	manoel@cs.umd.edu
Meny, Fred	Dept. of Commerce	fmeny@doc.gov
Milbank, Sam	AlliedSignal T S	samuel.milbank@alliedsignal.com
Miller, Glenn	Space Telescope Science Inst.	miller@stsci.edu
Miller, Roger N	General Dynamics	rmiller@gdeb.com
Minor, Susan	NASA/HQ	sminor@hq.nasa.gov
Moleski, Walt	NASA/GSFC	moleski@kong.gsfc.nasa.gov
Morasca, Sandro	Univeristy of Maryland	morasca@cs.umd.edu
Morisio, Maurizio	University of Maryland	morisio@cs.umd.edu
Murphy, Hugh	AlliedSignal	hmurphy@v2pop.hst.nasa.gov
Myers, Philip I	Computer Sciences Corp	pmyers@cscmail.csc.com
Nakano, Tetsuya	NASA/GSFC	tnakano@pop500.gsfc.nasa.gov
Narula, Nishi	OAO Corp	nnarula.aoa.com
Newton, Wally	Computer Sciences Corp	wbnewton@hotmail.com
Nichols, David	Jet Propulsion Lab	david.a.nichols@jpl.nasa.gov
Noone, Estelle	Computer Sciences Corp	enoone@csc.com
Norcio, Tony F	University of Maryland-BCO	norcio@umbc.edu
O'Donnell, Charlie	ECA, Inc	[No Email Address]
O'Neill, Don	Consultant	oneilldon@aol.com
O'Reilly, Frank	Northrop Grumman	oreilfr1@mail.northgrum.com
Page, Gerald T	Computer Sciences Corp	gpage@csc.com
Pajerski, Rose	Fraunhofer Center-Maryland	pajerski@fc-md.umd.edu
Panlilio-Yap, Nikki M	IBM	niknak@erols.com
Paquin, Sherry	Litton Marine Systems	sap01@cho.litton-marine.com
Parizer, Michael S	SATC	mparizer@pop300.gsfc.nasa.gov

Parra, Amy T	Computer Sciences Corp	aparra@csc.com
Patton, K. Kay	Computer Sciences Corp	kpatton@csc.com
Pavnica, Paul	Treasury – FinCEN	[No Email Address]
Phillips, William G	ARINC, Inc	bphillip@arinc.com
Pisano, Jim	National Radio Astronomy	jpisano@nrao.edu
Pitman, Andrew	Rowan University	ajp@torch.rowan.edu
Pittarelli, Ernie	Computer Sciences Corp	epittare@csc.com
Potter, Marshall R	DoD ODUSD (S&T) IS	pottermr@acg.osd.mil
Pradeep, Cheriyath	Raytheon Systems Co	cpradeep@eos.hitc.com
Ramamurty, Geeta	Computer Sciences Corp	gramamur@csc.com
Ray, Debasish	Nichols Advanced Marine	rayd@nichols.com
Regardie, Myrna L	Computer Sciences Corp	mregard@erols.com
Rifkin, Stan	Master Systems, Inc	sr@master-systems.com
Riley, Tom	NASA/GSFC	[No Email Address]
Rodgers, Thomas M	Lockheed Martin Corp	thomas.m.rodgers@lmco.com
Rohr, John A	Jet Propulsion Lab	john.a.rohr@jpl.nasa.gov
Rombach, H.Dieter	FhG IESE	rombach@iese.fhg.de
Rosenberg, Linda H	SATC Unisys	linda.rosenberg@gsfc.nasa.gov
Roy, Dan M	STP&P	danroy@stpp.com
Rus, Ioana	Fraunhofer Center-Maryland	irus@fc-md.umd.edu
Russell, Gabriella	Dept. of Commerce	grussell@doc.gov
Ryan, Charles J	SEI	ryan@sei.cmu.edu
Schneider, Laurie	Price Waterhouse	laurie.schneider@us.pwcglobal.com
Schulmeyer, Gordon G	PYXIS Systems Inter, Inc	pyxisinc@erols.com
Schultz, David J	Computer Sciences Corp	dschultz@csc.com
Scott, Hester	ALTA Systems, Inc	hscottcricochet.net
Seablom, Michael S	HSTX	seablom@gsfc.nasa.gov
Seaman, Carolyn B	University Maryland-Baltimore	cseaman@umbc.edu
Shami, Souha	QSS/GSFC/NASA	souha.r.shami@gsfc.nasa.gov
Sharma, Jagdish	NOAA	jagdish.sharma@noaa.gov
Shaw, Richard A	STSeI	shaw@stsei.edu
Shull, Forrest	Fraunhofer Center-Maryland	fshull@fraunhofer.org
Silver, Aaron N	Raytheon Systems Co	ansilver@west.raytheon.com
Small, Donald	AlliedSignal TS	donald.small@gsfc.nasa.gov
Smith, George F	Consultant	smithg@interpath.com
Smith, James A	NASA/GSFC	jasmith@hemlock.gsfc.nasa.gov
Smith, Sharon	DoD	[No Email Address]
Smith, Vivian A	FAA	vivian.smith@faa.gov
Spaulding, Omar	NASA/HQ	ospauldi@hq.nasa.gov
Spencer, Todd	Computer Sciences Corp	tspences@cscmail.csc.com
Squires, Burton E	Consultant	squiresb@acm.org

Stapko, Ruth	SATC	rstapko@pop300.gsfc.nasa.gov
Stark, Michael	NASA/GSFC	mstark@cs.umd.edu
Steinberg, Sandee	Computer Sciences Corp	ssteinbe@cscmail.csc.com
Steingrimsson, Borkur	McMaster University	borkur@mcserg.cas.mcmaster.ca
Straitt, Robert	USAFR	[No Email Address]
Swann, Mark H	U.S. Air Force	mark.swann@robins.af.mil
Swope, Janice	Computer Sciences Corp	[No Email Address]
Sykes, Mari	Computer Sciences Corp	msykes@cscmail.csc.com
Szulewski, Paul A	Mercury Computer Co	paulski@mc.com
Tervo, Betsy	Computer Sciences Corp	etervo@cscmail.csc.com
Tesoriero, Roseanne	Catholic University	tesoriero@cua.edu
Thomas, Bill	The MITRE Corp	bthomas@mitre.org
Tilley, Michael S	Raytheon ITSS	mtilley@stx.com
Travassos, Guilherme	University of Maryland	travasso@cs.umd.edu
Trimble, John	Howard University	trimble@scs.howard.edu
Tvedt, John	Catholic University	tvedt@cua.edu
Valente, Jr., Eduardo	Global Science & Tech	valente@gst.com
Valett, Jon	Q-Labs, Inc.	jon.valett@q-labs.com
Varney, Doug	OAO Corp	dvarney@oao.com
Vint, John	Northrop Grumman	vintjo@mail.northgrum.com
Vorndran, Ken	AlliedSignal	vorndrk@lskmp004.arsc.allied.com
Wahlberg, Melvyn	Computer Sciences Corp	mwahlber@cscmail.csc.com
Walker, Jocelyn	OAO Corp	jwalker@oao.com
Wallace, Dolores R	NIST	dwallace@nist.gov
Walter, Stephen O	Computer Sciences Corp	stevewalter@bigfoot.com
Wang, Alex S	Raytheon ITSS	awang@farside.gsfc.nasa.gov
Webby, Richard G	Telcordia Technologies	webby@research.telcordia.com
Webster, Bruce	Ntl. Cntr. for Environmental Prediction	bwebster@ncep.noaa.gov
Weller, Edward	Bull HN Information SYS	weller@bull.com
Wells, William	Computer Sciences Corp	wwells@cscmail.csc.com
Weszka, Joan	Lockheed Martin	joan.weszka@lmco.com
Wetzel, Paul E	Marconi	wetzelp@hotmail.com
Whisenand, Tom	Goldeg-Beacom College	whisent@goldeg.gbc.edu
Willey, Allan L	Motorola Labs	willey@motorola.com
Williams, Chadd	University of Maryland	chadd@cs.umd.edu
Wilson, Robert K	Jet Propulsion Lab	robert.k.wilson@jpl.nasa.gov
Wong, Eric	Telcordia Technologies	ewong@research.telcordia.com
Wortman, Kristin	Computer Sciences Corp	wortman@lheavx.gsfc.nasa.gov
Wynne, Denise	EDS	denise.wynne@nastech.eds.com
Yakimovich, Daniil	University of Maryland	dyak@cs.umd.edu
Zalesak, Steven T	NASA/GSFC	zalesak@gondor.gsfc.nasa.gov

Zavage, Jerry	Computer Sciences Corp	gzavage@csc.com
Zelkowitz, Marv	University of Maryland	mvz@cs.umd.eduw
Zero, Jose	University CA-Berkeley	zero@llnl.gov
Ziyad, Nigel	NASA/GSFC	ziyad@kong.gsfc.nasa.gov

Appendix B – Standard Bibliography of SEL Literature

The technical papers, memorandums, and documents listed in this bibliography are organized into two groups. The first group is composed of documents issued by the Software Engineering Laboratory (SEL) during its research and development activities. The second group includes materials that were published elsewhere but pertain to SEL activities. The *Annotated Bibliography of Software Engineering Laboratory Literature* contains an abstract for each document and is available via the SEL Products Page at http://sel.gsfc.nasa.gov/doc-st/docs/bibannot/contents.htm.

SEL-ORIGINATED DOCUMENTS

SEL-76-001, *Proceedings From the First Summer Software Engineering Workshop,* August 1976

SEL-77-002, *Proceedings From the Second Summer Software Engineering Workshop,* September 1977

SEL-78-005, *Proceedings From the Third Summer Software Engineering Workshop,* September 1978

SEL-78-006, *GSFC Software Engineering Research Requirements Analysis Study,* P. A. Scheffer and C. E. Velez, November 1978

SEL-78-007, *Applicability of the Rayleigh Curve to the SEL Environment,* T. E. Mapp, December 1978

SEL-78-302, *FORTRAN Static Source Code Analyzer Program (SAP) User's Guide (Revision 3),* W. J. Decker, W. A. Taylor, et al., July 1986

SEL-79-002, *The Software Engineering Laboratory: Relationship Equations,* K. Freburger and V. R. Basili, May 1979

SEL-79-004, *Evaluation of the Caine, Farber, and Gordon Program Design Language (PDL) in the Goddard Space Flight Center (GSFC) Code 580 Software Design Environment,* C. E. Goorevich, A. L. Green, and W. J. Decker, September 1979

SEL-79-005, *Proceedings From the Fourth Summer Software Engineering Workshop,* November 1979

SEL-80-002, *Multi-Level Expression Design Language-Requirement Level (MEDL-R) System Evaluation,* W. J. Decker and C. E. Goorevich, May 1980

SEL-80-005, *A Study of the Musa Reliability Model,* A. M. Miller, November 1980

SEL-80-006, *Proceedings From the Fifth Annual Software Engineering Workshop,* November 1980

SEL-80-007, *An Appraisal of Selected Cost/Resource Estimation Models for Software Systems,* J. F. Cook and F. E. McGarry, December 1980

SEL-80-008, *Tutorial on Models and Metrics for Software Management and Engineering*, V. R. Basili, 1980

SEL-81-011, *Evaluating Software Development by Analysis of Change Data*, D. M. Weiss, November 1981

SEL-81-012, *The Rayleigh Curve as a Model for Effort Distribution Over the Life of Medium Scale Software Systems*, G. O. Picasso, December 1981

SEL-81-013, *Proceedings of the Sixth Annual Software Engineering Workshop*, December 1981

SEL-81-014, *Automated Collection of Software Engineering Data in the Software Engineering Laboratory (SEL)*, A. L. Green, W. J. Decker, and F. E. McGarry, September 1981

SEL-81-101, *Guide to Data Collection*, V. E. Church, D. N. Card, F. E. McGarry, et al., August 1982

SEL-81-110, *Evaluation of an Independent Verification and Validation (IV&V) Methodology for Flight Dynamics*, G. Page, F. E. McGarry, and D. N. Card, June 1985

SEL-81-305, *Recommended Approach to Software Development*, L. Landis, S. Waligora, F. E. McGarry, et al., June 1992

SEL-81-305SP1, *Ada Developers' Supplement to the Recommended Approach*, R. Kester and L. Landis, November 1993

SEL-82-001, *Evaluation of Management Measures of Software Development*, G. Page, D. N. Card, and F. E. McGarry, September 1982, vols. 1 and 2

SEL-82-004, *Collected Software Engineering Papers: Volume 1*, July 1982

SEL-82-007, *Proceedings of the Seventh Annual Software Engineering Workshop*, December 1982

SEL-82-008, *Evaluating Software Development by Analysis of Changes: The Data From the Software Engineering Laboratory*, V. R. Basili and D. M. Weiss, December 1982

SEL-82-102, *FORTRAN Static Source Code Analyzer Program (SAP) System Description (Revision 1)*, W. A. Taylor and W. J. Decker, April 1985

SEL-82-105, *Glossary of Software Engineering Laboratory Terms*, T. A. Babst, M. G. Rohleder, and F. E. McGarry, October 1983

SEL-82-1306, *Annotated Bibliography of Software Engineering Laboratory Literature,* D. Kistler, J. Bristow, and D. Smith, November 1994

SEL-83-001, *An Approach to Software Cost Estimation*, F. E. McGarry, G. Page, D. N. Card, et al., February 1984

SEL-83-002, *Measures and Metrics for Software Development*, D. N. Card, F. E. McGarry, G. Page, et al., March 1984

SEL-83-003, *Collected Software Engineering Papers: Volume II*, November 1983

SEL-83-007, *Proceedings of the Eighth Annual Software Engineering Workshop*, November 1983

SEL-83-106, *Monitoring Software Development Through Dynamic Variables (Revision 1)*, C. W. Doerflinger, November 1989

SEL-84-003, *Investigation of Specification Measures for the Software Engineering Laboratory (SEL)*, W. W. Agresti, V. E. Church, and F. E. McGarry, December 1984

SEL-84-004, *Proceedings of the Ninth Annual Software Engineering Workshop*, November 1984

SEL-84-101, *Manager's Handbook for Software Development (Revision 1)*, L. Landis, F. E. McGarry, S. Waligora, et al., November 1990

SEL-85-001, *A Comparison of Software Verification Techniques*, D. N. Card, R. W. Selby, Jr., F. E. McGarry, et al., April 1985

SEL-85-002, *Ada Training Evaluation and Recommendations From the Gamma Ray Observatory Ada Development Team*, R. Murphy and M. Stark, October 1985

SEL-85-003, *Collected Software Engineering Papers: Volume III*, November 1985

SEL-85-004, *Evaluations of Software Technologies: Testing, CLEANROOM, and Metrics*, R. W. Selby, Jr. and V. R. Basili, May 1985

SEL-85-005, *Software Verification and Testing*, D. N. Card, E. Edwards, F. McGarry, and C. Antle, December 1985

SEL-85-006, *Proceedings of the Tenth Annual Software Engineering Workshop*, December 1985

SEL-86-001, *Programmer's Handbook for Flight Dynamics Software Development*, R. Wood and E. Edwards, March 1986

SEL-86-002, *General Object-Oriented Software Development*, E. Seidewitz and M. Stark, August 1986

SEL-86-003, *Flight Dynamics System Software Development Environment (FDS/SDE) Tutorial*, J. Buell and P. Myers, July 1986

SEL-86-004, *Collected Software Engineering Papers: Volume IV*, November 1986

SEL-86-005, *Measuring Software Design*, D. N. Card et al., November 1986

SEL-86-006, *Proceedings of the Eleventh Annual Software Engineering Workshop*, December 1986

SEL-87-001, *Product Assurance Policies and Procedures for Flight Dynamics Software Development*, S. Perry et al., March 1987

SEL-87-002, *Ada® Style Guide (Version 1.1)*, E. Seidewitz et al., May 1987

SEL-87-003, *Guidelines for Applying the Composite Specification Model (CSM)*, W. W. Agresti, June 1987

SEL-87-004, *Assessing the Ada® Design Process and Its Implications: A Case Study*, S. Godfrey, C. Brophy, et al., July 1987

SEL-87-009, *Collected Software Engineering Papers: Volume V*, November 1987

SEL-87-010, *Proceedings of the Twelfth Annual Software Engineering Workshop*, December 1987

SEL-88-001, *System Testing of a Production Ada Project: The GRODY Study*, J. Seigle, L. Esker, and Y. Shi, November 1988

SEL-88-002, *Collected Software Engineering Papers: Volume VI*, November 1988

SEL-88-003, *Evolution of Ada Technology in the Flight Dynamics Area: Design Phase Analysis*, K. Quimby and L. Esker, December 1988

SEL-88-004, *Proceedings of the Thirteenth Annual Software Engineering Workshop*, November 1988

SEL-88-005, *Proceedings of the First NASA Ada User's Symposium*, December 1988

SEL-89-002, *Implementation of a Production Ada Project: The GRODY Study*, S. Godfrey and C. Brophy, September 1989

SEL-89-004, *Evolution of Ada Technology in the Flight Dynamics Area: Implementation/ Testing Phase Analysis*, K. Quimby, L. Esker, L. Smith, M. Stark, and F. McGarry, November 1989

SEL-89-005, *Lessons Learned in the Transition to Ada From FORTRAN at NASA/Goddard*, C. Brophy, November 1989

SEL-89-006, *Collected Software Engineering Papers: Volume VII*, November 1989

SEL-89-007, *Proceedings of the Fourteenth Annual Software Engineering Workshop*, November 1989

SEL-89-008, *Proceedings of the Second NASA Ada Users' Symposium*, November 1989

SEL-89-103, *Software Management Environment (SME) Concepts and Architecture (Revision 1)*, R. Hendrick, D. Kistler, and J. Valett, September 1992

SEL-89-301, *Software Engineering Laboratory (SEL) Database Organization and User's Guide (Revision 3)*, L. Morusiewicz, February 1995

SEL-90-001, *Database Access Manager for the Software Engineering Laboratory (DAMSEL) User's Guide*, M. Buhler, K. Pumphrey, and D. Spiegel, March 1990

SEL-90-002, *The Cleanroom Case Study in the Software Engineering Laboratory: Project Description and Early Analysis*, S. Green et al., March 1990

SEL-90-003, *A Study of the Portability of an Ada System in the Software Engineering Laboratory (SEL)*, L. O. Jun and S. R. Valett, June 1990

SEL-90-004, *Gamma Ray Observatory Dynamics Simulator in Ada (GRODY) Experiment Summary*, T. McDermott and M. Stark, September 1990

SEL-90-005, *Collected Software Engineering Papers: Volume VIII*, November 1990

SEL-90-006, *Proceedings of the Fifteenth Annual Software Engineering Workshop*, November 1990

SEL-91-001, *Software Engineering Laboratory (SEL) Relationships, Models, and Management Rules*, W. Decker, R. Hendrick, and J. Valett, February 1991

SEL-91-003, *Software Engineering Laboratory (SEL) Ada Performance Study Report*, E. W. Booth and M. E. Stark, July 1991

SEL-91-004, *Software Engineering Laboratory (SEL) Cleanroom Process Model*, S. Green, November 1991

SEL-91-005, *Collected Software Engineering Papers: Volume IX*, November 1991

SEL-91-006, *Proceedings of the Sixteenth Annual Software Engineering Workshop*, December 1991

SEL-91-102, *Software Engineering Laboratory (SEL) Data and Information Policy (Revision 1)*, F. McGarry, August 1991

SEL-92-001, *Software Management Environment (SME) Installation Guide*, D. Kistler and K. Jeletic, January 1992

SEL-92-002, *Data Collection Procedures for the Software Engineering Laboratory (SEL) Database*, G. Heller, J. Valett, and M. Wild, March 1992

SEL-92-003, *Collected Software Engineering Papers: Volume X*, November 1992

SEL-92-004, *Proceedings of the Seventeenth Annual Software Engineering Workshop*, December 1992

SEL-93-001, *Collected Software Engineering Papers: Volume XI*, November 1993

SEL-93-002, *Cost and Schedule Estimation Study Report*, S. Condon, M. Regardie, M. Stark, et al., November 1993

SEL-93-003, *Proceedings of the Eighteenth Annual Software Engineering Workshop*, December 1993

SEL-94-001, *Software Management Environment (SME) Components and Algorithms*, R. Hendrick, D. Kistler, and J. Valett, February 1994

SEL-94-003, *C Style Guide*, J. Doland and J. Valett, August 1994

SEL-94-004, *Collected Software Engineering Papers: Volume XII*, November 1994

SEL-94-005, *An Overview of the Software Engineering Laboratory*, F. McGarry, G. Page, V. R. Basili, et al., December 1994

SEL-94-006, *Proceedings of the Nineteenth Annual Software Engineering Workshop*, December 1994

SEL-94-102, *Software Measurement Guidebook (Revision 1)*, M. Bassman, F. McGarry, R. Pajerski, June 1995

SEL-95-001, *Impact of Ada in the Flight Dynamics Division at Goddard Space Flight Center*, S. Waligora, J. Bailey, M. Stark, March 1995

SEL-95-003, *Collected Software Engineering Papers: Volume XIII*, November 1995

SEL-95-004, *Proceedings of the Twentieth Annual Software Engineering Workshop*, December 1995

SEL-95-102, *Software Process Improvement Guidebook (Revision 1),* K. Jeletic, R. Pajerski, C. Brown, March 1996

SEL-96-001, *Collected Software Engineering Papers: Volume XIV*, October 1996

SEL-96-002, *Proceedings of the Twenty-First Annual Software Engineering Workshop*, December 1996

SEL-97-001, Guide To Software Engineering Laboratory Data Collection And Reporting, September 1997

SEL-97-002, *Collected Software Engineering Papers: Volume XV*, October 1997

SEL-97-003, *Proceedings of the Twenty-Second Annual Software Engineering Workshop*, December 1997

SEL-98-001, *SEL COTS Study Phase 1 - Initial Characterization Study Report*, A. Parra, August 1998

SEL-98-002, Proceedings of the Twenty-Third Annual Software Engineering Workshop*, December 1998*

SEL-99-001A, Profile of Software at the Information Systems Center, December 1999

SEL-99-002, Proceedings of the Twenty-fourth Annual Software Engineering Workshop*, December 1999*

SEL-00-001, *Collected Software Engineering Papers: Volume XVI*, March 2000

SEL-00-002, *Collected Software Engineering Papers: Volume XVII*, March 2000

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	November 1999	Technical Memorandum

4. TITLE AND SUBTITLE

Proceedings of the Twenty-Fourth Annual Engineering Workshop

5. FUNDING NUMBERS

Code 581
IDIQ 5-2857-G

6. AUTHOR(S)

Compiled by GSFC.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS (ES)

Goddard Space Flight Center
Greenbelt, Maryland 20771

8. PEFORMING ORGANIZATION REPORT NUMBER

2000–01799-0

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS (ES)

National Aeronautics and Space Administration
Washington, DC 20546-0001

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

CP—2000–209890

11. SUPPLEMENTARY NOTES

This work was performed under the auspices of IDIQ 5-2857-G.

12a. DISTRIBUTION / AVAILABILITY STATEMENT

Unclassified–Unlimited
Subject Category:61
Report available from the NASA Center for AeroSpace Information,
7121 Standard Drive, Hanover, MD 21076-1320. (301) 621-0390.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

On December 1 and 2, the Software Engineering Laboratory (SEL), a consortium composed of NASA/Goddard, the University of Maryland, and CSC, held the 24th Software Engineering Workshop (SEW), the last of the millennium. Approximately 240 people attended the 2-day workshop.

Day 1 was composed of four sessions: International Influence of the Software Engineering Laboratory; Object Oriented Testing and Reading; Software Process Improvement; and Space Software

For the first session, three internationally known software process experts discussed the influence of the SEL with respect to software engineering research. In the Space Software session, prominent representatives from three different NASA sites—GSFC's Marti Szczur, the Jet Propulsion Laboratory's Rick Doyle, and the Ames Research Center IV&V Facility's Lou Blazy—discussed the future of space software in their respective centers. At the end of the first day, the SEW sponsored a reception at the GSFC Visitors' Center.

Day 2 also provided four sessions: Using the Experience Factory; A panel discussion entitled "Software Past, Present, and Future: Views from Government, Industry, and Academia"; Inspections; and COTS

The day started with an excellent talk by CSC's Frank McGarry on "Attaining Level 5 in CMM Process Maturity." Session 2, the panel discussion on software, featured NASA Chief Information Officer Lee Holcomb (Government), our own Jerry Page (Industry), and Mike Evangelist of the National Science Foundation (Academia). Each presented his perspective on the most important developments in software in the past 10 years, in the present, and in the future.

14. SUBJECT TERMS

SEL, software engineering, SEW, object-oriented, software process improvement, SPI, CMM, Goal-Question-Metric, GQM, experience factory, EF, IV&V, software inspections, COTS, COTS integration, UML, space mission software, autonomous missions, future software, JINI.

15. NUMBER OF PAGES

570

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL